STATISTICAL METHODS WITH APPLICATIONS TO MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

A Thesis
Presented to
The Academic Faculty

by

Yibiao Lu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology
August 2012

STATISTICAL METHODS WITH APPLICATIONS TO MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Approved by:

Professor Xiaoming Huo, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Shi-Jie Deng
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Ming Yuan
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Alex Shapiro
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Panagiotis Tsiotras
The Daniel Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: Apr 26, 2012

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis consists of four chapters. The first chapter is on function estimation via regularization. The second and third chapters are about static/dynamic path planning algorithms, respectively. The last chapter is application-oriented, focusing on spikes prediction in electricity prices.

Chapter 1 focuses on theoretical results on high-order laplacian-based regularization in function estimation. Thin-plate splines are widely used for estimating an unknown function in nonparametric regression, but it performs poorly over complex regions (i.e., regions with hole) and therefore soap-film smoother is proposed which introduces the laplacian-based regularization and allows for certain degree of freedom along the boundary. Besides, in machine learning community, unsupervised learning algorithm such as laplacian eigenmap also adopts the graphical laplacian regularization as a control for unsmoothness of the underlying manifold. However, the theoretical justification for Laplacian-based regularization in terms of optimal convergence rate has been missing in the literatures. In Chapter 1, we study the iterated laplacian regularization in the context of supervised learning in order to achieve both nice theoretical properties (like thin-plate splines) and good performance over complex region (like soap film smoother). We first derive a closed-form function estimation in the context of nonparametric smoothing based on the graphical laplacian regularization. It is shown that soap film is one special case of this smoother when the order of penalty is 2. We then prove that the smoother achieves the optimal rate of convergence. The essential part of proof is to study the asymptotic properties of the penalty matrix's eigenvalues, which relies on the Sobolev semi-norms, spectrum analysis of elliptic operator and the relationship between heat kernel and laplacian

operator. The last part of chapter 1 is devoted to the proof of asymptotic optimality of tuning parameter λ selected by generalized cross-validation (GCV).

In Chapter 2, we propose an innovative static path-planning algorithm called *m-A** within an environment full of obstacles, which has lower worst-case order magnitude of computation complexity and reduces the number of vertex expansion compared to the benchmark A* algorithm in the simulation study. More specifically, the proposed graph structure first employs a *recursive dyadic partitioning* to divide the environment into "block" of different sizes. The block sizes are determined by the relative importance of information within those blocks. The preprocessing of information within each block is handled by an innovative *bottom-up fusion* algorithm, which accumulates distance information from finer scale to coarser scale, therefore we only need the boundary vertices in each block to conduct the shortest path planning (In other words, we obtain a sparsity representation of the environment). We then show that modified A* based on beamlet graphical structure reduces the number of vertex expansion from $O(n^2 \log(n))$ to $O(n^2)$, when the environment is a $n \times n$ image. In the simulation study, our approach outperforms A* armed with both standard L_1 heuristic and stronger ones such as True-Distance heuristics (TDH), yielding faster query time, adequate usage of memory and reasonable preprocessing time. More generally, the recursive dyadic partitioning scheme and bottom-up fusion algorithm do not rely on A* algorithm and therefore can be adapted to any other path-planning approach that involves multiscale strategy.

Chapter 3 proposes m-LPA* algorithm that extends the m-A* algorithm proposed in Chapter 2 in the context of dynamic path-planning and shows that m-LPA* achieves better performance compared to the benchmark: lifelong planning A* (LPA*) in terms of robustness and worst-case computational complexity. Employing the same beamlet graphical structure as m-A*, m-LPA* encodes the information of the environment in a hierarchical, multiscale fashion, keeping track of "long-range"

interaction between the vertices in the beamlet graph. When the update of original graph induces "local dead-end", which causes the surprisingly huge number of vertex expansion in the replanning, the information is transmitted by modifying the hierarchical structure of beamlet graph, so no more "dead-ends" exist in the new graph, and hence it produces a more robust dynamic path-planning algorithm. The analysis of computational complexity reveals that, in the worst case, the proposed algorithm has a lower order of complexity than the LPA* algorithm. In our numerical experiments, it shows that the m-LPA* algorithm can dramatically reduce the number of vertex expansions in the worst scenarios.

Chapter 4 focuses on an approach for the prediction of spot electricity spikes via a combination of boosting and wavelet analysis. It has been well recognized in finance that modeling spikes (i.e., jumps) is an essential task in asset pricing, risk management and trading activity. Due to the complexity and uncertainties in the power grid, spot electricity prices are highly volatile and normally carry with spikes, which may be tens or even hundreds of times higher than the normal price. Another issue comes from modeling the intraday seaonality and its evolution over time. The first part of our proposed scheme considers the hourly spot prices within one day as a high dimensional vector and applys the nondecimated wavelet analysis to detect the spikes for training. The second part utilizes the gradient boosting trees for the spikes prediction with carefully selected predictors from Australian electricity markets. Extensive numerical experiments show that our approach improves the prediction accuracy, thanks to the fact that the gradient boosting trees method inherits the good properties of decision trees such as robustness to the irrelevant covariates, fast computational capability and good interpretation.

CHAPTER I

THEORETICAL RESULTS ON HIGH-ORDER LAPLACIAN-BASED REGULARIZATION IN FUNCTION ESTIMATION

1.1 Introduction

Numerous approaches have been proposed to solve the data smoothing problem, which emerges in applications from engineering, bioinformatics, and finance. Given noisy observations, the standard approach is to introduce penalty term on the smoothness of the underlying function and make a trade-off between the goodness-of-fit and the penalty. The penalty term typically involves the estimation of underlying function's derivatives, and Frobenius norm of Hessian is definitely the classic choice, which is introduced by [21] and [74] under the name *Thin-plate Splines*. [74] established that the thin-plate estimator achieves the best convergence rate for nonparametric estimator in terms of L_2 risk according to [66]. From the computational point of view, the tuning parameter selected by generalized cross validation can be proved to be the optimal choice.

The limitation of thin-plate splines on its capability of handling irregular regions is discussed in [61], which leads to the *soap film* smoother proposed in [75]. The regularization term employed in soap film is actually the integral of squared laplacian instead of Frobenius norm of Hessian, and therefore admits certain degree of freedom along the boundary of region and therefore it can handle data smoothing over irregular regions. However, it has some drawbacks: *first*, soap film smoothers performs poorly over regular domain in some cases, which certainly restricts its applicability; *second*, current version of the soap film smoothers lacks theoretical justification in terms

of consistency and whether or not it achieves the optimal convergence rate; *third*, soap film smoother involves the solution of PDE's, namely Laplacian equation and Poisson equation, which is not common in machine learning community and it is computationally inefficient in the high dimensional case.

Iterated laplacian regularization makes its first appearance in [78], and has the laplacian-based regularization employed in soap film as its special case when the order $m = 2$. Its corresponding discrete approximation is based on *graphical laplacian* used in [5] to capture the local smoothness of underlying manifold. [78] shows that the discrete approximation actually converges to the iterated laplacian regularization and furthermore provides some intuition according to the theory of reproducing kernel Hilbert space. Still, the theoretical justification of iterated laplacian regularization in terms of convergence rate is not established, unlike the thin-plate splines.

In this chapter, we consider the data smoothing problem using least square loss function and iterated laplacian regularization, and name the corresponding estimator as *Graphical Laplacian Smoother (GLS)*. We establish the optimal rate of convergence for GLS, and therefore provide the first appearance of theoretical justification for laplacian-based regularization in parallel to the one for Hessian-based regularization utilized in thin-plate splines. As in [78], we summarized the difference between thin-plate splines and iterated laplacian regularization: iterated laplacian regularization can be viewed as a generalization of the thin plate splines from regular domains to unknown submanifolds, from a coordinate dependent Sobolev semi-norm defined by partial derivatives to a coordinate free iterated Laplacian semi-norm using Laplacians, from fixed data independent reproducing kernels to data dependent kernels. Our proof also shows the intuition why soap film method could smooth data over complex regions: the approximation of iterated laplacian by graphical laplacian actually involves the Gaussian kernel and it essentially provides local regularization, which could handle the irregular regions. Besides, our proof explains the reason why

soap film doesn't work as expected under certain circumstances due to the violation of boundary condition required to achieve optimal rate of convergence. Furthermore, we establish the asymptotic optimality of generalized cross validation for GLS, which gives a justifiable way of choosing tuning parameter. This is also missing from the previous works.

The rest of the paper is organized as follows. Section 1.2 describes our proposed regularization term and computational approach. Section 1.3 studied the rates of eigenvalues of the power of discrete graphical laplacian matrix and proved that the iterated laplacian regularization reaches the optimal convergence rate of nonparametric smoothing.

1.2 Methodology

In this section, we will consider the *iterated laplacian* as the new regularization term in the context of supervised learning. Some general notations adopted in the smoothing splines will be introduced in Section 1.2.1. In Section 1.2.2, iterated laplacian is defined and the corresponding computational scheme is illustrated in details. Section 1.2.3 describes the GCV (*generalized cross validation*) approach for choosing the optimal penalty parameter λ.

1.2.1 Notations

Let observations be $(X_i, y_i), i = 1, 2, ..., n$ and assume that the data generation mechanism is

$$y_i = f(X_i) + \varepsilon_i$$

where y_i's are responses and $X_i \in \mathbb{R}^d$ are the predictors. The capitalized X_i indicates that it can be multivariate. Function estimation is to undercover $f(\cdot)$. To search for a function $\hat{f}$ in a functional space $\mathcal{F}$ such that it is a reasonable estimate of the true

underlying function f, the standard approach is to minimize a functional as follows:

$$\min_{f\in\mathcal{F}} \sum_{i=1}^{n} (y_i - f(X_i))^2 + \lambda \mathcal{J}(f) \tag{1}$$

where the first term is called a goodness-of-fit measure and the second term penalizes the unsmoothness. The above optimization can be considered as the trade-off between the estimation error and model complexity. We consider the functional space $\mathcal{F}$ to be a Sobolev space. More specifically, let $\mathbb{Z}_+^d$ denote the set of all ordered d-tuples of nonnegative integers. For $\alpha \in \mathbb{Z}_+^d$, $\alpha = (\alpha_1, \alpha_2, ..., \alpha_d)$ and $|\alpha| = \sum_{i=1}^d \alpha_i$. Define the partial derivative

$$D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_d^{\alpha_d}}$$

Then the Sobolev space of order m, denoted by $W^{m,2}(\Omega)(= H^m(\Omega))$ is defined to be the space consisting of those functions in $L^2(\Omega)$ that, together with all their weak partial derivatives up to and including those of order m, belong to $L^2(\Omega)$, i.e., we have

$$H^m(\Omega) = \left\{ f : D^\alpha f \in L^2(\Omega), \forall \alpha \in \mathbb{Z}_+^d, |\alpha| \le m \right\}$$

where $\Omega \in \mathbb{R}^d$ is the domain of the function.

Define Sobolev semi-inner product

$$\langle u, v \rangle_m = \sum_{|\alpha|=m} \binom{m}{\alpha} \int_\Omega D^\alpha u D^\alpha v dx \tag{2}$$

The induced Sobolev semi-norm (defined in [74]) is

$$J_m^d(f) = \sum_{|\alpha|=m} \binom{m}{\alpha} \int_\Omega |D^\alpha f|^2 dx = \sum_{|\alpha|=m} \binom{m}{\alpha} \|D^\alpha f\|_{L^2}^2 \tag{3}$$

and $J_m^d(\cdot)$ is shown to be a good choice in the sense that the estimated function f is continuous and lies in the corresponding RKHS (Reproducing Kernel Hilbert Space) if $2m > d$. It has nice theoretical foundation rooted in functional analysis and theory of Sobolev space. Difference between many smoothers lies in the choice of regularization term, namely $\mathcal{J}(f)$. We will discuss our choice, the *iterated laplacian regularization*, in the following section.

1.2.2 Problem Formulation

Throughout this chapter, we assume the uniformly sampled $T = \{X_i\}_{i=1}^n$. Our formulation of the problem becomes

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2 + \lambda I_m^d(f) \tag{4}$$

where $I_m^d(f) = |f|_{\Omega,m}^2 = \int_\Omega f(x)\Delta^m f(x)dx$. Note that $\Delta = -\sum_{i=1}^d \frac{\partial^2}{\partial x_i^2}$.

We proposed the discrete approximation to the term $I_m^d(f)$ as follows.

$$I_{m,n}^d(f) = \frac{1}{n}\left(\frac{1}{nt_n^{d/2+1}}\right)^m \mathbf{f}^T\mathbf{L}^m\mathbf{f} \tag{5}$$

where $\mathbf{f} = (f(X_1), ..., f(X_n))^T$, $\mathbf{L} = \mathbf{D} - \mathbf{W}$. $\mathbf{D}$ is a diagonal matrix with diagonal element $D_{ii} = \sum_j w_{ij}$ and $\mathbf{W} = (w_{ij})_{1\le i,j\le n}$, where $w_{ij} = e^{-\frac{\|x_i - x_j\|^2}{t}}$, t is the bandwidth that depends on n. $\mathbf{L}$ is called *Graphical Laplacian* matrix in [4]. The structure of w_{ij} is essentially the gaussian kernel which imposes the local relation that depends on the bandwidth t.

This approximation is justified by Theorem 1.2.1 from [78],

Theorem 1.2.1 *Let Ω be a compact connected submanifold in $\mathbb{R}^d$ without boundary. $x_1, ..., x_n$ are sampled uniformly on Ω, and m be a positive integer. Assume $f \in C^{2m}(\Omega), \mathrm{Vol}(\Omega) = 1$, then for $t_n = O(n^{-\frac{1}{d+2+\alpha}})$ where $\alpha > 0$ as $n \to \infty$ we have*

$$\frac{c}{n}\left(\frac{1}{nt_n^{d/2+1}}\right)^m \mathbf{f}^T\mathbf{L}^m\mathbf{f} \xrightarrow{p} \int_\Omega f(x)\Delta^m f(x)dx = I_m^d(f) \tag{6}$$

where t_n is the bandwidth of a Gaussian kernel function and c is a constant.

Therefore, our discrete approximation to (4) is as follows

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2 + \lambda I_{m,n}^d(f) \tag{7}$$

which is equivalent to

$$\min_{f\in\mathcal{F}} (\mathbf{y}-\mathbf{f})^T(\mathbf{y}-\mathbf{f}) + \frac{\lambda}{n^m t^{m(d/2+1)}}\mathbf{f}^T\mathbf{L}^m\mathbf{f} \tag{8}$$

Define $\mathbf{M} = (nt^{d/2+1})^{-m}\mathbf{L}^m$, then to study the optimality of our estimation $\hat{\mathbf{f}} = (I + \lambda\mathbf{M})^{-1}\mathbf{y}$, which is obtained from first order condition of optimization (8), is equivalent to the study of eigenvalues of matrix $\mathbf{M}$, which will be fully examined in Section 1.3.

1.2.3 Choice of the Penalty Parameter λ

We adopted the generalized cross validation (GCV) to determine the penalty parameter λ. The generalized cross validation function is defined as

$$\mathrm{GCV}_n(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_{n,\lambda}(X_i)}{1 - \frac{1}{n}\mathrm{tr}[\mathbf{A_n}(\lambda)]}\right)^2$$

where $\mathbf{A}_n(\lambda) = (\mathbf{I}_n + \lambda\mathbf{M})^{-1}$. The optimal value of the penalty parameter λ can be estimated by minimizing the above GCV function, i.e.,

$$\hat{\lambda}_G = \mathrm{argmin}_{\lambda>0}\mathrm{GCV}_n(\lambda)$$

For practice purpose, we could re-parameterize the $\lambda = e^\theta$, $\theta \in \mathbb{R}$ to convert the minimization into a unconstrained optimization problem. The justification is relatively straightforward and can be found in [39].

1.3 Theoretical Properties

From this point on, the estimator established in Section 1.2 will be named the GLS (graphical laplacian smoothing). Section 1.3.1 provides the necessary foundation, mainly, some inequalities from functional analysis, as the preparation for the study of optimal convergence rate. In Section 1.3.2, we discuss the asymptotic rates of penalty matrix $\mathbf{M}$'s eigenvalues using the tools developed in Section 1.3.1. The multivariate input situation ($d > 1$) is included in Section 1.3.3, where we show that asymptotic properties are comparable to those of thin-plate splines in [74]. Finally we introduce the asymptotic optimality of the GCV, and show in Section 1.3.4 that for GLS, the asymptotic optimality of GCV is preserved, hence GCV is a justifiable way of choosing parameter λ.

1.3.1 Mathematical Preparation

From Section 1.2, we pin down our problem to bound the eigenvalues of penalty matrix $\mathbf{M}$. Since $\mathbf{M} \propto \mathbf{L}^m$ and L is positive semi-definite symmetric matrix, we will first consider the bound of eigenvalues of $\mathbf{M}$ defined in Section 1.2.2 when $m = 1$, then extend it to the case of $m > 1$.

For any domain $\Omega \subset \mathbb{R}^d$ which satisfies Lemma 1.3.1 below, let $H^m(\Omega)$ denote the mth-order Sobolev space of generalized functions. Then define the semi-inner product by

$$\langle f, g \rangle_{\Omega,m} = \int_\Omega f(x)\Delta^m g(x)dx = \int_\Omega g(x)\Delta^m f(x)dx \tag{9}$$

which gives rise to the related semi-norm

$$|f|^2_{\Omega,m} = \int_\Omega f(x)\Delta^m f(x)dx \tag{10}$$

Lemma 1.3.1 *$|f|_{\Omega,m}$ is a semi-norm given one of the following two conditions:*

1. $\partial\Omega = \emptyset$
2. $\nabla(\Delta^k f(x)) \cdot \boldsymbol{n} = 0, \forall x \in \partial\Omega, k = 0, 1, ..., m-1.$

where $\partial\Omega$ denotes the boundary of Ω. ∇ stands for gradient operator and $\mathbf{n}$ is the normal vector orthogonal to $\partial\Omega$.

The first condition requires the empty boundary of Ω, which corresponds to the *closed submanifold* in Euclidean space. In this chapter, we focus on the second case. This requirement of boundary is quite similar to *Neumann Boundary Condition* (also appeared in [61]), which has a nice physical meaning (see Section 1.5.3 for further illustration.). The proof of Lemma 1.3.1 bases on the Green's identity and it can be found in Section 1.5.

With $T = \{X_i\}_{i=1}^n$, we can also define a discrete version of the aforementioned semi-norm as

$$|f|^2_{T,m} = \frac{1}{n}\sum_{i=1}^n f(X_i)\Delta^m f(X_i) \tag{11}$$

and when $m = 0$, $|f|_{T,0}^2 = \frac{1}{n}\sum_{i=1}^n f(X_i)^2$ and $|f|_{\Omega,0}^2 = \int_\Omega f^2(x)dx$. Let $\mathbf{E}_{T,1}$ be the representing matrix such that $|f|_{T,1}^2 = \frac{1}{n}\mathbf{f}^T\mathbf{E}_{T,1}\mathbf{f}$, where $\mathbf{f} = (f(X_1), ..., f(X_n))^T$ and f corresponds to the solution of a variational problem:

$$|f|_{T,1} = \min_{\phi \in H^1(\Omega), \phi(X_i)=y_i} |\phi|_{T,1} \tag{12}$$

The existence of matrix $E_{T,1}$ is established as follows. Please refer to Section 1.5 for detailed proof.

Proposition 1.3.2 *There exists a matrix $\mathbf{E}_{T,m}$, such that*

$$|f|_{T,m}^2 = \min_{\phi \in H^m(\Omega), \phi(X_i)=y_i} |\phi|_{T,m}^2 = \frac{1}{n} y^T \mathbf{E}_{\mathbf{T},\mathbf{m}} y \tag{13}$$

where $T = \{X_i\}_{i=1}^n$, $\mathbf{y} = (y_1, y_2, .., y_n)^T = (f(X_1), f(X_2), ..., f(X_n))^T$.

For the set of sampling points $T = \{X_i\}_{i=1}^n$ in domain Ω, we assume that there exists a constant $B_0 > 0$ such that $\delta_{\max}/\delta_{\min} \leq B_0$, where $\delta_{\max} = \sup_{X\in\Omega} \inf_{X_i \in T} \|X - X_i\|$, and $\delta_{\min} = \min_{j\neq i} \|X_j - X_i\|$. Next we establish some properties needed for domain Ω. The proof is included in Section 1.5.

Lemma 1.3.3 *If Ω is a bounded domain in $\mathbb{R}^d$ and satisfies the condition in Lemma 1.3.1. Denote e_n as the largest eigenvalue of matrix $\boldsymbol{E}_{T,1}$, then $n\delta_{\max}^d$ and $\delta_{\max}^2 e_n$ are both bounded from above, i.e., there exists constant B_1, B_2 such that $n\delta_{\max}^d \leq B_1$ and $\delta_{\max}^2 e_n \leq B_2$.*

Recall that a domain with Lipschitz boundary is a set in Euclidean space whose boundary is sufficiently regular in the sense that it can be thought of as locally being the graph of a Lipschitz continuous function. Let Ω be an open set in $\mathbb{R}^d$ satisfying a *uniform cone condition*, which is defined as follows.

Definition An open set Ω in $\mathbb{R}^d$ is said to satisfy the uniform cone condition, if there exists a radius $r > 0$ and an angle $\theta \in (0, \pi/2)$ such that for any $X \in \Omega$, a unit vector

$\zeta(X) \in \mathbb{R}^d$ exists such that the cone

$$C(X, \zeta(X), r, \theta) = \{X + t\mathbf{s} : \mathbf{s} \in \mathbb{R}^d, \|\mathbf{s}\| = 1, \zeta(X)^T\mathbf{s} \geq \cos\theta, 0 \leq t \leq r\}$$

contains in Ω.

To exhibit the Rayleigh quotient inequalities connecting Sobolev semi-norms (same as our definition when $m = 1$) and their discretized version. Let

$$U_2^1(\Omega) = \{f \in H^1(\Omega) : \underline{B}|f|_{\Omega,1}^2 \leq |f|_{T,1}^2 \leq \bar{B}|f|_{\Omega,1}^2\}$$

be a class of functions with bilaterally bounded constraint on their first order derivative, where $\underline{B}, \bar{B}$ are independent of f. We have the following result with respect to $U_2^1(\Omega)$ and the proof can be found in Section 1.5.

Lemma 1.3.4 *The functional class $U_2^1(\Omega)$ has at least a subset as the polynomial spline space of degree $m+1$ with knots at $T = \{X_i\}_{i=1}^n$.*

Notice that $|f|_{\Omega,1}^2 = \int_\Omega f(x)\Delta f(x)dx = \int_\Omega \|\nabla f(x)\|^2 dx = \int_\Omega \sum_{|\alpha|=1} |D^\alpha f(x)|^2 dx$ under conditions in Lemma 1.3.1, and this is the same as the classic definition of semi-norm in Sobolev space. In order to examine the connection between the continuous and discrete version of semi-norms, we have the following lemmas, which are key steps to establish the closeness of eigenvalues of $\mathbf{E}_{T,1}$ and spectrals of elliptic operator Δ.

Lemma 1.3.5 *Let Ω be an open bounded Lipschitz domain satisfying both uniform cone condition and conditions in Lemma 1.3.1. Then there exists constant $C_1 = C_1(d, \Omega, B_0, \underline{B}) > 0$ and $\delta_0 > 0$ such that if $\delta_{\max} \leq \delta_0$, we have*

$$\frac{|f|_{T,1}^2}{|f|_{T,0}^2} \geq \frac{|f|_{\Omega,1}^2}{C_1(|f|_{\Omega,0}^2 + \delta_{\max}^2|f|_{\Omega,1}^2)} \tag{14}$$

Lemma 1.3.6 *Let Ω be an open bounded Lipschitz domain satisfying both uniform cone condition and conditions in Lemma 1.3.1. Then there exists constant $C_2 = C_2(d, \Omega, B_0, \underline{B}, \bar{B}) > 0$ and $\delta_0 > 0$ such that if $\delta_{\max} \leq \delta_0$, we have*

$$\frac{|f|_{\Omega,1}^2}{|f|_{\Omega,0}^2} \geq \frac{|f|_{T,1}^2}{C_2(|f|_{T,0}^2 + \delta_{\max}^2|f|_{T,1}^2)} \tag{15}$$

Let $e_1 \leq \cdots \leq e_n$ be the eigenvalues of $\mathbf{E}_{T,1}$ in ascending order. Clearly $\{e_j\}_{j=1}^n$ are non-negative real numbers since the matrix $\mathbf{E}_{T,1}$ is semi-positive definite. Next we will establish the convergence rate of the eigenvalues and show that they can be bounded by the discrete spectrum of the iterated laplacian Δ^1.

Lemma 1.3.7 *Let Ω be an open bounded Lipschitz domain satisfying both uniform cone condition and conditions in Lemma 1.3.1. Then there exists constants $C_3, C_4 > 0$ such that*

$$C_3\rho_j \leq e_j \leq C_4\rho_j$$

where $\rho_1 \leq \rho_2 \leq ... \leq \rho_n$ are the first n eigenvalues of the variational eigenvalue problem

$$\langle \phi, \psi \rangle_{\Omega,1} = \rho \langle \phi, \psi \rangle_{\Omega,0} \quad \forall \psi \in H_2^1(\Omega)$$

Based on the above lemmas, we have one of our main results:

Theorem 1.3.8 *Let Ω be an open bounded Lipschitz domain satisfying both uniform cone condition and conditions in Lemma 1.3.1. Recall $e_1 \leq e_2 \leq ... \leq e_n$ are the eigenvalues of $\boldsymbol{E}_{T,1}$ in ascending order. Then there exists constants $C_5, C_6 > 0$ such that for $1 < j \leq n$, we have*

$$C_5 j^{2/d} \leq e_j \leq C_6 j^{2/d}$$

Proof Apply Theorem 14.6 in [1] to get $\rho_j \sim j^{2/d}$ if $j > 1$. And with result from Lemma 1.3.7, it concludes the proof.

We include Theorem 14.6 of [1] in the appendix for completeness. Here we require $j > 1$ because for Laplacian operator Δ, its smallest eigenvalue $\rho_1 = 0$ and the corresponding eigenfunction ϕ_1 spans the null space of this operator. Correspondingly, it is obvious that $\mathbf{L}$ also has 0 as its eigenvalue and vector $\mathbf{1}$ as the corresponding eigenvector.

1.3.2 Bounds of regularization matrix M's eigenvalues

The basic idea is simple: since we have already studied the eigenvalues of matrix $\mathbf{E}_{T,1}$, and if we can build connection between matrix $\mathbf{L}$ and $\mathbf{E}_{T,1}$ (w.r.t. $m = 1$), the bound of $\mathbf{M}(\propto \mathbf{L}^m)$'s eigenvalues is then quite obvious.

In order to achieve this, we first list some properties of matrix $\mathbf{L}$:

1. $\mathbf{L} = \mathbf{L}^T$.

2. $\mathbf{L} \succeq 0$ and the smallest eigenvalue equals 0. It is the discrete approximation of $-\Delta$ operator.

3. $\mathbf{L} = \frac{1}{2}\sum_{i=1}^{n} \mathbf{L}_i$, where

$$\mathbf{L}_i = \begin{pmatrix} w_{i,1} & 0 & 0 & -w_{i,1} & 0 & 0 & 0 \\ 0 & \ddots & 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & w_{i,i-1} & -w_{i,i-1} & 0 & 0 & 0 \\ -w_{i,1} & \cdots & -w_{i,i-1} & \sum_{j\neq i} w_{ij} & -w_{i,i+1} & \cdots & -w_{i,n} \\ 0 & \cdots & 0 & -w_{i,i+1} & w_{i,i+1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -w_{i,n} & 0 & \cdots & w_{i,n} \end{pmatrix}$$

The mathematical intuition of defining $\mathbf{L}_i$ is that it uses all sampled data points with kernel function to approximate the laplacian around X_i. It becomes more clear if we consider

$$\mathbf{L}_i\mathbf{f} = (w_{i1}f_1 - w_{i1}f_i, ..., \sum_{j:j\neq i} w_{ij}f_i - \sum_{j:j\neq i} w_{ij}f_j, ..., w_{in}f_n - w_{in}f_i)^T$$

and the following lemmas. The proofs are quite similar to those from [5], which considers more general form on manifolds. Here we consider the case in the Euclidean space.

Lemma 1.3.9 *Given any open ball $B \subset \Omega$ and $p \in B$, for any $l \in \mathbb{N}$, it holds that as $t \to 0$,*

$$\int_B e^{-\frac{\|p-y\|^2}{4t}} f(y)dy - \int_\Omega e^{-\frac{\|p-y\|^2}{4t}} f(y)dy = o(t^l) \tag{16}$$

Lemma 1.3.9 proves the fact that we can use only a small open set (i.e. the Euclidean ball in our case) around a fixed data point to estimate the heat kernel over Ω at that point, and the error decays exponentially as the bandwidth shrinks. This will be useful when we establish the local approximation of the laplacian operator.

Lemma 1.3.10 *There exists a constant C such that*

$$\frac{\partial}{\partial t}\left((4\pi t)^{-\frac{d}{2}} \int_{B(p)} e^{-\frac{\|p-y\|^2}{4t}} f(y)dy\right)\Bigg|_{t=0} = \Delta f(p) + Cf(p) \tag{17}$$

Lemma 1.3.10 established the connection between the heat kernel and the laplacian operator. Its proof is the same as that of Lemma 9 in [5]. The combination of Law of Large Numbers (LLN) and the following Lemma 1.3.11 shows that the Laplacian at each data point can be approximated by weighted average of function values at sampled data points, which will be used to establish the asymptotic properties of matrix $\mathbf{M}$.

Lemma 1.3.11

$$\lim_{t\to 0}(\pi t)^{-d/2-1}\left(\int_\Omega w_{ij} f(x_i)dx_j - \int_\Omega w_{ij} f(x_j)dx_j\right) = \Delta f(x_i)$$

where $w_{ij} = e^{-\frac{\|x_i - x_j\|^2}{t}}$ is the value of gaussian kernel between x_i and x_j.

Finally we arrives another main result:

Theorem 1.3.12 *Let $\mu_1 \leq \cdots \leq \mu_n$ be the eigenvalues of the matrix $\boldsymbol{M}$. There exists constants $C_7, C_8 > 0$ such that for $1 < j \leq n$ we have*

$$C_7 j^{2m/d} \leq \mu_j \leq C_8 j^{2m/d}.$$

1.3.3 Convergence Rate of Multivariate GLS Estimator

In this section, we will establish the optimal rate of convergence for GLS estimation, based on the asymptotic properties of matrix **M**'s eigenvalues. First we need the following lemma:

Lemma 1.3.13 *If for $B_3, B_4 > 0$, we have $B_3 j^m \leq \mu_j \leq B_4 j^m$ for a constant $m > 0$ and $j = 1, 2, ...$, then the following holds for $n > 0, \lambda > 0$,*

$$\sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2} = O(\lambda^{-\frac{1}{m}}) \tag{18}$$

Proof of the above lemma is in Appendix. The following theorem is the main result in this section:

Theorem 1.3.14 *Let $\hat{\boldsymbol{f}}_n(\lambda) = \boldsymbol{A}_n(\lambda) = (I_n + \lambda \boldsymbol{M})^{-1}\boldsymbol{y}$ be the estimator of the laplacian regularizer with the order $m > d/2$ and denote $r_n(\lambda) = n^{-1}\|\hat{\boldsymbol{f}}_n(\lambda) - \boldsymbol{f}\|^2$. If $n \to \infty$ and $\lambda \sim n^{-2m/(2m+d)}$ is chosen, then*

$$\mathbb{E}[r_n(\lambda)] = O(n^{-\frac{2m}{2m+d}}) \tag{19}$$

In particular, if the smoothing parameter is chosen to satisfy $\lambda \sim n^{-2m/(2m+d)}$, we achieve the convergence rate $\mathbb{E}[r_n(\lambda)] = O(n^{-\frac{2m}{2m+d}})$, which is the optimal convergence rate for multivariate function estimation with the order m in d-dimensional space ([66]).

1.3.4 Asymptotic Optimality of GCV

In this section, we will show that our proposed estimator satisfies some general conditions and then prove the asymptotic optimality of GCV under our proposed framework.

1.3.4.1 General Conditions

Let $\hat{\mathbf{f}}_n(\lambda) = \mathbf{A_n}(\lambda)\mathbf{y} = (\mathbf{I_n} + \lambda\mathbf{M})^{-\mathbf{1}}\mathbf{y}$ be the estimator from smoothing splines of iterated laplacian regularization with order m and denote $r_n(\lambda) = n^{-1}\|\hat{\mathbf{f}}_n(\lambda) - \mathbf{f}\|^{\mathbf{2}}$. The asymptotic optimality of GCV is defined as

$$\frac{r_n(\hat{\lambda}_G)}{\inf_{\lambda\in\mathbb{R}_+} r_n(\lambda)} \xrightarrow{p} 1 \tag{20}$$

which verifies the closeness between the values of risk function given by the GCV choice $\hat{\lambda}_G$ and theoretically optimal choice λ^*, where $\lambda^* = \arg\inf_{\lambda\in\mathbb{R}_+} r_n(\lambda)$.

The main result of this section is to show that our estimator satisfies the following three conditions.

(A.1) $\inf_{\lambda\in\mathbb{R}_+} n\mathbb{E}[r_n(\lambda)] \to \infty$.

(A.2) There exists a sequence $\{\lambda_n\}$ such that $r_n(\lambda_n) \xrightarrow{p} 0$.

(A.3) Let $0 \le \kappa_1 \le ... \le \kappa_n$ be the eigenvalues of $\mathbf{K_n}(\lambda) = \lambda\mathbf{M}$. For any l such that $l/n \to 0$, then as $n \to \infty$,

$$\frac{(n^{-1}\sum_{i=l+1}^{n}\kappa_i^{-1})^2}{n^{-1}\sum_{i=l+1}^{n}\kappa_i^2} \to 0$$

From Theorem 1.3.12, we already know that $\mu_1 = 0$ and denote the null space spanned by the first eigenfunction of Δ as $\mathcal{N}$. Then we have the following, which provides the verification of condition (A.1).

Lemma 1.3.15 *If $f \notin \mathcal{N}$, the estimator $\hat{\mathbf{f}}_n(\lambda)$ has the property: $\inf_{\lambda\in\mathbb{R}_+} n\mathbb{E}[r_n(\lambda)] \to \infty$, which is (A.1).*

In order to establish the bounds on errors, it would be convenient if the random term $r_n(\lambda) = n^{-1}\|\hat{\mathbf{f}}_n(\lambda) - \mathbf{f}\|^2$ can be replaced by its expectation, which is deterministic. Fortunately, it is true and we state it in the following Lemma:

Lemma 1.3.16 *Under condition (A.1), we have $\sup_{\lambda>0}\left|\frac{r_n(\lambda)}{E[r_n(\lambda)]} - 1\right| \to 0$.*

The condition (A.2) shows that the risk function $r_n(\lambda_n)$ converges to zero in probability with proper sequence $\{\lambda_n\}$. From Theorem 1.3.14, we know $\mathbb{E}[r_n(\lambda)] \to 0$ as $n \to \infty$, if $\lambda \sim n^{-\frac{2m}{2m+d}}$. Besides, $r_n(\lambda)$ and its expectation are "close" according to Lemma 1.3.16, therefore (A.2) holds true.

Again since $\mu_j = O(j^{2m/d})$, we have the following lemma.

Lemma 1.3.17 *In our model, for any l such that $l/n \to 0$ and $\kappa_{l+1} > 0$, the ratio*

$$\frac{(n^{-1}\sum_{i=l+1}^{n}\kappa_i^{-1})^2}{n^{-1}\sum_{i=l+1}^{n}\kappa_i^2} \to 0, \qquad as \quad n \to \infty.$$

This verifies the condition (A.3) and is an intermediate result used in Section 1.3.4.2 and it plays an important role in the asymptotic analysis. Its proof is provided in Section 1.5.

1.3.4.2 Asymptotic Optimality Theorem

Under the aforementioned three conditions ((A.1)-(A.3)), we will prove the asymptotic optimality of GCV.

Lemma 1.3.18 *Under the condition (A.2), we have*

$$n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)] \to 1, \tag{21}$$

and

$$n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2 \to \sigma^2. \tag{22}$$

The asymptotic results in Lemma 1.3.18 will be used in the proof of Lemma 1.3.20.

Lemma 1.3.19 *Under the condition (A.3), for a sequence λ_n such that $r_n(\lambda_n) \to 0$, we have*

$$\frac{(n^{-1}\mathrm{tr}[\mathbf{A_n}(\lambda_\mathbf{n})])^\mathbf{2}}{n^{-1}\mathrm{tr}[\mathbf{A_n}(\lambda_\mathbf{n})^\mathbf{2}]} \to 0. \tag{23}$$

Finally we build the connection between $r_n(\lambda)$ and $\mathrm{SURE}_n(\lambda)$, and its proof can be found in Section 1.5.

Lemma 1.3.20 *For any $\hat{\lambda}$ such that $r_n(\lambda) \to 0$ and*

$$\frac{(n^{-1}\mathrm{tr}[\mathbf{A_n}(\hat{\lambda})])^{\mathbf{2}}}{n^{-1}\mathrm{tr}[\mathbf{A_n}(\hat{\lambda})^{\mathbf{2}}]} \to 0 \tag{24}$$

under the condition (A.1), we have

$$\frac{\left|\mathrm{SURE}_n(\hat{\lambda}) - \tilde{r}_n(\hat{\lambda}) - n^{-1}\|\varepsilon\|^2 + \sigma^2\right|}{r_n(\hat{\lambda})} \xrightarrow{p} 0, \tag{25}$$

and

$$\frac{n^{-1}\|\tilde{\mathbf{f}}_n(\hat{\lambda}) - \hat{\mathbf{f}}_n(\hat{\lambda})\|^2}{r_n(\hat{\lambda})} \xrightarrow{p} 0, \tag{26}$$

where

$$\begin{aligned}
\mathrm{SURE}_n(\lambda) &= \sigma^2 - \sigma^4 \frac{(n^{-1}\mathrm{tr}[\mathbf{I_n} - \mathbf{A_n}(\lambda)])^{\mathbf{2}}}{n^{-1}\|(\mathbf{I_n} - \mathbf{A_n}(\lambda))\mathbf{y}\|^{\mathbf{2}}}, \\
\tilde{\mathbf{f}}_n(\lambda) &= \mathbf{y} - \sigma^{\mathbf{2}} \frac{\mathrm{tr}[\mathbf{I_n} - \mathbf{A_n}(\lambda)]}{\|(\mathbf{I_n} - \mathbf{A_n}(\lambda))\mathbf{y}\|^{\mathbf{2}}}(\mathbf{I_n} - \mathbf{A_n}(\lambda))\mathbf{y}, \\
\tilde{r}_n(\lambda) &= n^{-1}\|\tilde{\mathbf{f}}_n(\lambda) - \mathbf{f}\|^{\mathbf{2}}.
\end{aligned}$$

Theorem 1.3.21 *Under condition (A.2) and (A.3), $\hat{\mathbf{f}}_n(\hat{\lambda}_G)$ is consistent, i.e., $r_n(\hat{\lambda}_G) \to 0$, where $\hat{\lambda}_G$ is chosen by GCV.*

The above theorem establishes the asymptotic consistency of GCV's choice of tuning parameter and its proof is provided in Section 1.5.

Theorem 1.3.22 *Under condition (A.1)-(A.3), $\hat{\mathbf{f}}_n(\hat{\lambda}_G)$ is asymptotically optimal, where $\hat{\lambda}_G$ is the GCV choice, i.e. $r_n(\hat{\lambda}_G)/r_n(\lambda_n^*) \xrightarrow{p} 1$, where λ_n^* is the best possible choice that is only known by oracles.*

This theorem finally established the asymptotic optimality of GCV.

1.4 Conclusion

In this chapter, we considered iterated laplacian regularization in context of supervised learning and established the theoretical foundation correspondingly. More

specifically, we proved that under realistic regulation conditions, our estimator achieves the optimal convergence rate. To this purpose, we studied the asymptotic behavior of eigenvalues of iterated laplacian matrix by making the connection between semi-norm in Sobolev space and spectrum analysis of elliptic operator. Besides we showed that generalized cross validation still provides the best tuning parameter in terms of consistency and optimality. Our estimator can be viewed as the generalization of both thin-plate splines and soap film smoothing (to higher order laplacian regularization) with nice theoretical justification.

1.5 Appendix

1.5.1 Detailed Proofs

Proof of Lemma 1.3.1

Proof Using Green's Identity, we have $|f|_{\Omega,m} \geq 0$ and

$$|f|_{\Omega,m} = \begin{cases} \int_\Omega |\Delta^{m/2} f(x)|^2 dx & \text{even} \quad m, \\ \int_\Omega |\nabla(\Delta^{(m-1)/2} f(x))|^2 dx & \text{odd} \quad m. \end{cases} \tag{27}$$

Proof of Proposition 1.3.2

Proof Step 1: we define $\|y\| = \left(\min_{\phi \in H^m(\Omega), \phi(X_i)=y_i} |\phi|^2_{T,m}\right)^{1/2}$ and show that $\|\cdot\|$ is a semi-norm. Therefore we verify the following three properties:

1. Clearly, for any $y \in \mathbb{R}^n$, $\|y\| \geq 0$.

2. For any ϕ such that $\phi(X_i) = y_i$, since $\lambda\phi(X_i) = \lambda y_i$, we have $\|\lambda y\| \leq |\lambda\phi|_{T,m} = |\lambda||\phi|_{T,m}$. Thus, $\|\lambda y\| \leq |\lambda|\|y\|$.

3. Triangle inequality. Assume $y = y^1 + y^2$. For any $\varepsilon > 0$, there exist ϕ_1, ϕ_2, such that

$$\|y^1\| \geq |\phi_1|_{T,m} - \varepsilon, \quad \|y^2\| \geq |\phi_2|_{T,m} - \varepsilon.$$

Then,

$$\|y^1\| + \|y^2\| \geq |\phi_1 + \phi_2|_{T,m} - 2\varepsilon.$$

Since we know that $(\phi_1 + \phi_2)(X_i) = y_i^1 + y_i^2 = y_i$ for $i = 1, 2, ..., n$, thus,

$$\|y^1\| + \|y^2\| \geq \|y\| - 2\varepsilon.$$

Let $\varepsilon \to 0$, we have $\|y^1\| + \|y^2\| \geq \|y\| = \|y^1 + y^2\|$.

Step 2: we show that $\|\cdot\|$ satisfies the following equality:

$$\|u + v\|^2 + \|u - v\|^2 = 2\|u\|^2 + 2\|v\|^2.$$

For $\forall \varepsilon > 0$, there exist ϕ_1, ϕ_2 s.t. $\phi_1(X_i) = u_i, \phi_2(X_i) = v_i, i = 1, 2, ..., n$ and $\|u\|^2 \geq |\phi_1|_{T,m}^2 - \varepsilon, \|v\|^2 \geq |\phi_2|_{T,m}^2 - \varepsilon$. Then,

$$\begin{aligned}
2\|u\|^2 + 2\|v\|^2 &\geq 2|\phi_1|_{T,m}^2 + 2|\phi_2|_{T,m}^2 - 4\varepsilon \\
&= 2\left|\frac{\phi_1 + \phi_2}{2} + \frac{\phi_1 - \phi_2}{2}\right|_{T,m}^2 + 2\left|\frac{\phi_1 + \phi_2}{2} - \frac{\phi_1 - \phi_2}{2}\right|_{T,m}^2 - 4\varepsilon \\
&= |\phi_1 + \phi_2|_{T,m}^2 + |\phi_1 - \phi_2|_{T,m}^2 - 4\varepsilon \\
&= \|u + v\|^2 + \|u - v\|^2 - 4\varepsilon.
\end{aligned}$$

Let $\varepsilon \to 0$, we have $\|u + v\|^2 + \|u - v\|^2 \leq 2\|u\|^2 + 2\|v\|^2$. Similarly, we replace u, v from above with $\frac{u+v}{2}$ and $\frac{u-v}{2}$, we get $\|u\|^2 + \|v\|^2 \leq \frac{1}{2}\|u + v\|^2 + \frac{1}{2}\|u - v\|^2$.

Step 3: we define a bilinear function based on $\|\cdot\|$:

$$\langle u, v\rangle \triangleq \frac{1}{4}\left(\|u + v\|^2 - \|u - v\|^2\right).$$

We only need to verify two properties, which are quite straightforward.

1. $\langle u_1 + u_2, v\rangle = \langle u_1, v\rangle + \langle u_2, v\rangle$.

2. $\langle \lambda u, v\rangle = \lambda\langle u, v\rangle$.

Step 4: For any fixed v, define $\psi_v : \mathbb{R}^n \to \mathbb{R}$, $\psi_v(u) = \langle u, v\rangle$. Since $\langle u, v\rangle$ is bilinear, ψ_v is linear, and therefore from Riesz representation theorem, there exists $w_v \in \mathbb{R}^n$, s.t. $\psi_v(u) = u^T w_v$.

Let $G : u \to w_v$. Clearly G is a linear mapping on $\mathbb{R}^n$. Hence there exists $E \in \mathbb{R}^{n\times n}$ s.t. $G(v) = Ev$ and $\langle u, v\rangle = u^T Ev$.

Since $\|u\|^2 = \langle u, u\rangle = u^T Eu$, E is semi-positive definite. Let $\mathbf{E}_{T,m} = nE$, we have

$$\frac{1}{n} y^T \mathbf{E}_{T,m} y = \langle y, y\rangle = \|y\|^2 = \min_{\phi\in H^m(\Omega),\phi(X_i)=y_i} |\phi|^2_{T,m}.$$

Proof of Lemma 1.3.3

Proof Let V_d be the volume of unit ball in $\mathbb{R}^d$, and then $nV_d\delta^d_{\min} \le \mathrm{Vol}(\Omega)$. Therefore, we have

$$\delta^d_{\max} \le n^{-1}\frac{\mathrm{Vol}(\Omega)}{V_d}\frac{\delta^d_{\max}}{\delta^d_{\min}} = n^{-1}\frac{\mathrm{Vol}(\Omega)}{V_d}B^d_0 = O(n^{-1}),$$

from which we get $n\delta^d_{\max}$ is bounded from above.

Next, let u be a function satisfies

$$\frac{e_n}{n} = \frac{1}{n}\mathbf{u}^T\mathbf{E}_{T,1}\mathbf{u} = |u|_{T,1} = \min_{\phi\in H^1_\Omega,\phi(X_i)=u_i} |\phi|_{T,1},$$

where $\mathbf{u} = (u_1, ..., u_n)$ is the eigenvector of $\mathbf{E}_{T,1}$ corresponding to e_n. We define a compactly supported radial basis function

$$w(s) = \begin{cases} e^{-\|s\|/(1-\|s\|)} & 0 \le \|s\| \le 1, \\ 0 & \|s\| > 1. \end{cases}$$

and specify $\phi(X) = \sum_{i=1}^n u_i w_i(X)$ where $w_i(X) = w(\frac{X-X_i}{\delta_{\min}})$. Clearly $\phi(X_i) = u_i$. Moreover, we have for any multi-index $\alpha \in \mathbb{Z}^d_+$

$$D^\alpha w_i(X_j) = 0 \quad \forall i \ne j,$$

and with $|\alpha| = 2$

$$D^\alpha w_j(X_j) = \delta^{-2}_{\min} D^\alpha w(0).$$

Hence, we have

$$\begin{aligned}
|u|_{T,1}^2 &\leq |\phi|_{T,1}^2 \\
&= \frac{1}{n}\sum_{j=1}^{n}\left(\phi(X_j)\Delta\phi(X_j)\right) \\
&= \frac{1}{n}\sum_{j=1}^{n}\left(\sum_i u_i w_i(X_j)\right)\Delta\left(\sum_i u_i w_i(X_j)\right) \\
&= \frac{1}{n}\sum_{j=1}^{n} u_j^2 w_j(X_j)\Delta w_j(X_j) \\
&\leq \frac{1}{n}\sum_{j=1}^{n} u_j^2\left(w_j(X_j)\sum_{|\alpha|=2}|D^\alpha w_j(X_j)|\right) \\
&\leq \frac{1}{n}\sum_{j=1}^{n} u_j^2\left(\sum_{|\alpha|=2}|D^\alpha w(0)|\right)\delta_{\min}^{-2},
\end{aligned}$$

which implies that $e_n \leq C(w)\delta_{\min}^{-2}$, where $C(w) = \sum_{|\alpha|=2}|D^\alpha w(0)|$.

Finally we get

$$\delta_{\max}^2 e_n \leq C(w)\frac{\delta_{\max}^2}{\delta_{\min}^2} = C(w)B_0^2$$

and prove that $\delta_{\max}^2 e_n$ is bounded from above.

Proof of Lemma 1.3.4

Proof Given $T = \{X_i\}_{i=1}^n$, we denote $\Pi : \{a = X_0 < X_1 < \cdots < X_n < X_{n+1} = b\}$ as the partition of $\Omega = [a, b]$. Let

$$\begin{aligned}
\mathcal{S}_2(\Pi) = \{s(t) : s(t) = s_i(t) \in \mathcal{P}_2, t \in [X_i, X_{i+1}], i = 0, 1, \cdots, n; \\
D^l s_{i-1}(X_i) = D^l s_i(X_i), i = 1, \cdots, n, l = 0, 1; s'(a) = s'(b) = 0\}
\end{aligned}$$

be a spline space of degree 2. The functional class $U_2^1(\Omega)$ is non-empty and it suffices to show that $\mathcal{S}_2(\Pi) \subset U_2^1(\Omega)$.

Obviously, $\mathcal{S}_2(\Pi) \subset \mathcal{W}_2^1(\Omega)$. For any $f \in \mathcal{S}_2(\Pi)$, assume $f(t)|_{[X_i,X_{i+1}]} = a_0 + a_1 t +$

a_2t^2, so we have

$$\begin{aligned}\frac{1}{2}[f'(X_i)^2+f'(X_{i+1})^2] &= a_1^2+2a_1a_2(X_i+X_{i+1})+4a_2^2\frac{X_i^2+X_{i+1}^2}{2},\\ \frac{1}{X_{i+1}-X_i}\int_{X_i}^{X_{i+1}}[f'(t)]^2dt &= a_1^2+2a_1a_2(X_i+X_{i+1})+4a_2^2\frac{X_i^2+X_iX_{i+1}+X_{i+1}^2}{3},\end{aligned}$$

From which we know $\frac{1}{2}[f'(X_i)^2+f'(X_{i+1})^2]\geq\frac{1}{X_{i+1}-X_i}\int_{X_i}^{X_{i+1}}[f'(t)]^2dt$. Besides,

$$\begin{aligned}\frac{1}{2}[f'(X_i)^2+f'(X_{i+1})^2] &\leq \frac{1}{2}[f'(X_i)^2+f'(X_{i+1})^2]+2\left(a_1+2a_2\frac{X_i+X_{i+1}}{2}\right)^2\\ &= 3[a_1^2+2a_1a_2(X_i+X_{i+1})]+4a_2^2\left(\frac{X_i^2+X_{i+1}^2}{2}+\frac{(X_i+X_{i+1})^2}{2}\right)\\ &= \frac{3}{X_{i+1}-X_i}\int_{X_i}^{X_{i+1}}[f'(t)]^2dt.\end{aligned}$$

On one hand, we have

$$\begin{aligned}|f|_{T,1} &= \frac{1}{2n}\sum_{i=0}^{n}\left([f'(X_i)]^2+[f'(X_{i+1})]^2\right)\\ &\geq \frac{1}{n}\sum_{i=0}^{n}\frac{1}{X_{i+1}-X_i}\int_{X_i}^{X_{i+1}}[f'(t)]^2dt\\ &\geq \frac{1}{n\delta_{\max}}|f|_{\Omega,1}^2\geq\frac{1}{B_0(b-a)}|f|_{\Omega,1}^2.\end{aligned}$$

On the other hand, we have

$$\begin{aligned}|f|_{T,1} &\leq \frac{1}{n}\sum_{i=0}^{n}\frac{3}{X_{i+1}-X_i}\int_{X_i}^{X_{i+1}}[f'(t)]^2dt\\ &\leq \frac{3}{n\delta_{\min}}|f|_{\Omega,1}^2\leq\frac{3B_0}{b-a}|f|_{\Omega,1}^2.\end{aligned}$$

Therefore, $\mathcal{S}_2(\Pi)\subset U_2^1(\Omega)$.

Proof of Lemma 1.3.5

Proof According to [73], there exists constant $C(d,m,\Omega,B_0)>0$ and $\delta>0$ such that for $\delta_{\max}\leq\delta_0$, we have

$$|f|_{T,0}^2\leq C(d,m,\Omega,B_0)\left(|f|_{\Omega,0}^2+\delta_{\max}^2|f|_{\Omega,1}^2\right).$$

Since $|f|_{T,1}^2 \geq \underline{B}|f|_{\Omega,1}^2$, we have

$$\frac{|f|_{T,1}^2}{|f|_{T,0}^2} \geq \frac{\underline{B}|f|_{\Omega,1}^2}{C(d,m,\Omega,B_0)\left(|f|_{\Omega,0}^2+\delta_{\max}^2|f|_{\Omega,1}^2\right)} = \frac{|f|_{\Omega,1}^2}{C_1\left(|f|_{\Omega,0}^2+\delta_{\max}^2|f|_{\Omega,1}^2\right)},$$

where $C_1 = C(d,m,\Omega,B_0)/\underline{B}$.

Proof of Lemma 1.3.6

Proof According to [73], there exists constant $C(d,m,\Omega,B_0) > 0$ and $\delta_0 > 0$ such that for $\delta_{\max} \leq \delta_0$, we have

$$|f|_{\Omega,0}^2 \leq C'(d,m,\Omega,B_0)\left(|f|_{T,0}^2+\delta_{\max}^2|f|_{T,1}^2\right).$$

Since $\underline{B}|f|_{\Omega,1}^2 \leq |f|_{T,1}^2 \leq \bar{B}|f|_{\Omega,1}^2$, we have

$$\frac{|f|_{\Omega,1}^2}{|f|_{\Omega,0}^2} \geq \frac{|f|_{T,1}^2/\bar{B}}{C'(d,m,\Omega,B_0)\left(|f|_{T,0}^2+\delta_{\max}^2|f|_{T,1}^2/\underline{B}\right)} \geq \frac{|f|_{T,1}^2}{C_2\left(|f|_{T,0}^2+\delta_{\max}^2|f|_{T,1}^2\right)},$$

where $C_2 = \bar{B}C'(d,m,\Omega,B_0)\max(1,1/\underline{B})$.

Proof of Lemma 1.3.7

Proof From Lemma 1.3.5, it holds

$$\frac{|\phi|_{T,1}^2}{|\phi|_{T,0}^2} \geq \frac{|\phi|_{\Omega,1}^2}{C_1\left(|\phi|_{\Omega,0}^2+\delta_{\max}^2|\phi|_{\Omega,1}^2\right)}$$

for any $\phi \in H^m(\Omega)$ with $|\phi|_{T,0}^2 \neq 0$. Then $e_j \geq \frac{1}{C_1}\theta_j$, where $\theta_1 \leq \cdots \leq \theta_n$ are the first n eigenvalues of the variational eigenvalue problem

$$|\phi|_{\Omega,1}^2 = \theta \cdot \left(|\phi|_{\Omega,0}^2+\delta_{\max}^2|\phi|_{\Omega,1}^2\right),$$

which implies $\theta_j = \frac{\rho_j}{1+\delta_{\max}^2\rho_j}$, for any $j = 1,\cdots,n$.

Note that $\delta_{\max}^2\rho_j$ is bounded from above, since $\rho_j \sim j^{\frac{2}{d}}$ according to Theorem 14.6 in [1] and the fact that $\delta_{\max}^2 = O(n^{-2/d})$ from Lemma 1.3.3. So there exists $C_3 > 0$ such that

$$\frac{1}{C_1(1+\delta_{\max}^2\rho_j)} \geq C_3,$$

then we have $e_j \geq C_3 \rho_j$.

On the other hand, using Lemma 1.3.6, we have

$$\frac{|\phi|^2_{\Omega,1}}{|\phi|^2_{\Omega,0}} \geq \frac{|\phi|^2_{T,1}}{C_1\left(|\phi|^2_{T,0} + \delta^2_{\max}|\phi|^2_{T,1}\right)},$$

which implies $\rho_j \geq \frac{1}{C_2}\nu_j$, where $\nu_1 \leq \cdots \leq \nu_n$ are the first n eigenvalues of the variational eigenvalue problem

$$|\phi|^2_{T,1} = \nu \cdot \left(|\phi|^2_{T,0} + \delta^2_{\max}|\phi|^2_{T,1}\right),$$

which gives

$$\nu_j = \frac{e_j}{1 + \delta^2_{\max} e_j}, \quad j = 1, \cdots, n.$$

So there exists $C_4 > 0$ such that

$$e_j \leq C_2(1 + \delta^2_{\max} e_j)\rho_j \leq C_2(1 + \delta^2_{\max} e_n)\rho_j \leq C_4 \rho_j,$$

since $\delta^2_{\max} e_n$ is bounded according to the Lemma 1.3.3.

Proof of Lemma 1.3.9

Proof Let $d = \inf_{x \notin B} \|p - x\|^2$ and let $B^c = \Omega - B$. From B is open and Ω is locally compact, we know $d > 0$. Thus,

$$\left| \int_B e^{-\frac{\|p-y\|^2}{4t}} f(y)dy - \int_\Omega e^{-\frac{\|p-y\|^2}{4t}} f(y)dy \right| \leq \mu(B^c) \sup_{x \in B^c} (|f(x)|) e^{-\frac{d^2}{4t}},$$

where $\mu(B^c)$ denotes the Lebesgue measure of set B^c. The first two terms are constant and $e^{-\frac{d^2}{4t}}$ approaches 0 faster than any polynomial as $t \to 0$.

Proof of Lemma 1.3.11

Proof From Lemma 1.3.10, if we treat $f(p)$ as a constant function, we have

$$\frac{\partial}{\partial t}\left((4\pi t)^{-\frac{d}{2}} \int_{B(p)} e^{-\frac{\|p-y\|^2}{4t}} f(p)dy\right)|_{t=0} = Cf(p). \tag{28}$$

Denote $A(t) = (4\pi t)^{-\frac{d}{2}} \int_\Omega e^{-\frac{\|p-y\|^2}{4t}} f(y)dy$. Using the property of heat kernel, we have

$$A(0) = \lim_{t\to 0}(4\pi t)^{-\frac{d}{2}} \int_{B(p)} e^{-\frac{\|p-y\|^2}{4t}} f(p)dy = f(p). \tag{29}$$

Therefore,

$$\begin{aligned}\Delta f(p) &= \lim_{t\to 0} \frac{A(t) - A(0)}{t} \\ &= \lim_{t\to 0}(4\pi t)^{-\frac{d}{2}} \left(\int_\Omega e^{-\frac{\|p-y\|^2}{4t}} f(p)dy - \int_\Omega e^{-\frac{\|p-y\|^2}{4t}} f(y)dy \right).\end{aligned}$$

This completes our proof by setting $p = x_i, y = x_j$.

Proof of Theorem 1.3.12

Proof Without loss of generality, let $\mathbf{f}$ be the vector of function values at the knots of $T = \{X_i\}_{i=1}^n$ normalized by $|f|_{T,0}^2 = \frac{1}{n}\mathbf{f}^T\mathbf{f} = 1$.

Then

$$\begin{aligned}\left| \frac{1}{nt^{d/2+1}} \frac{\mathbf{f}^T\mathbf{L}\mathbf{f}}{\mathbf{f}^T\mathbf{f}} - \frac{|f|_{T,1}^2}{|f|_{T,0}^2} \right| &= \left| \frac{1}{n^2 t^{d/2+1}} \mathbf{f}^T\mathbf{L}\mathbf{f} - |f|_{T,1}^2 \right| \\ &= \left| \frac{1}{n^2 t^{d/2+1}} \sum_{i=1}^n f(x_i) \sum_{j:j\neq i} (w_{ij}f(x_i) - w_{ij}f(x_j)) - \frac{1}{n}\sum_{i=1}^n f(x_i)\Delta f(x_i) \right| \\ &\leq \frac{M}{n} \sum_{i=1}^n \left| \frac{1}{nt^{d/2+1}} \sum_{j:j\neq i} (w_{ij}f(x_i) - w_{ij}f(x_j)) - \Delta f(x_i) \right|,\end{aligned}$$

where M is the upper bound of $|f(X)|$ on Ω.

Define $Z = \frac{1}{n}\sum_{j:j\neq i} (w_{ij}f_i - w_{ij}f_j)$, and we know

$$\mathbb{E}[Z] = f(x_i) \int_\Omega w_{ij}dx_j - \int_\Omega w_{ij}f(x_j)dx_j.$$

Also from Hoeffding's Inequality,

$$\mathbb{P}\left(\frac{1}{t^{d/2+1}} |Z - \mathbb{E}[Z]| \geq \varepsilon \right) \leq e^{-\varepsilon^2 n t^{d+2}}.$$

Therefore,

$$\left|\frac{1}{t^{d/2+1}}Z-\Delta f(x_i)\right| \le \underbrace{\left|t^{-\frac{d+2}{2}}(Z-\mathbb{E}[Z])\right|}_{(I)} + \underbrace{\left|t^{-\frac{d+2}{2}}\left(\int_\Omega w_{ij}f(x_i)dx_j-\int_\Omega w_{ij}f(x_j)dx_j\right)-\Delta f(x_i)\right|}_{(II)}.$$

Since

$$(I)\le e^{-\varepsilon^2 nt^{d+2}}t^{-\frac{d+2}{2}}|Z-\mathbb{E}[Z]|+(1-e^{-\varepsilon^2 nt^{d+2}})\varepsilon\to 0,$$

as $n\to\infty$ if $t_n=O(n^{-\frac{1}{d+2+\alpha}})$ where $\alpha>0$.

Besides, from Lemma 1.3.11, we know $(II)\to 0$ as $n\to\infty$. Therefore all the above implies $|\mu_j-e_j|\to 0$ as $n\to\infty$. Hence from Theorem 1.3.8, we have

$$C_7 j^{2/d}\le\mu_j\le C_8 j^{2/d}.$$

For the case $m>1$, we have the following clearly

$$C_7 j^{2m/d}\le\mu_j\le C_8 j^{2m/d}.$$

Then our proof is completed.

Proof of Lemma 1.3.13

Proof Evidently we have

$$\sum_{j=1}^m\frac{1}{(1+\lambda B_2 j^m)^2}\le\sum_{j=1}^n\frac{1}{(1+\lambda\mu_j)^2}\le\sum_{j=1}^n\frac{1}{(1+\lambda B_1 j^m)^2}.$$

For $i=1,2$, we have

$$\begin{aligned}\sum_{j=1}^n\frac{1}{(1+\lambda B_i j^m)^2}&\ge\int_1^{n+1}\frac{1}{(1+\lambda B_i x^m)^2}dx\\&=\frac{1}{m}\int_{\lambda B_i}^{\lambda B_i(n+1)^m}\frac{y^{-\frac{m-1}{m}}}{(1+y)^2}dy\cdot(\lambda B_i)^{-\frac{1}{m}}\\&\to m^{-1}\left(\int_{\lambda B_i}^\infty\frac{y^{-\frac{m-1}{m}}}{(1+y)^2}dy\right)B_i^{-1/m}\cdot\lambda^{-1/m}\\&=O(\lambda^{-1/m}),\end{aligned}$$

where the second equation comes from the change of variable $y = \lambda B_i x^m$. Similarly we also have the following

$$\begin{aligned}
\sum_{j=1}^{n} \frac{1}{(1+\lambda B_i j^m)^2} &\leq \int_0^n \frac{1}{(1+\lambda B_i x^m)^2} dx \\
&= \frac{1}{m} \int_0^{\lambda B_i n^m} \frac{y^{-\frac{m-1}{m}}}{(1+y)^2} dy \cdot (\lambda B_i)^{-\frac{1}{m}} \\
&\to m^{-1} \left(\int_{\lambda B_i}^{\infty} \frac{y^{-\frac{m-1}{m}}}{(1+y)^2} dy \right) B_i^{-1/m} \cdot \lambda^{-1/m} \\
&= O(\lambda^{-1/m}).
\end{aligned}$$

Proof of Theorem 1.3.14

Proof By using the bounds of eigenvalues of matrix $\mathbf{M}$ that $\mu_j = O(j^{\frac{2m}{d}})$ obtained from Theorem 1.3.12, we have

$$\begin{aligned}
\mathbb{E}[r_n(\lambda)] &= \mathbb{E}[n^{-1} \|\hat{\mathbf{f}}_n(\lambda) - \mathbf{f}\|^2] \\
&= n^{-1} \left(\mathbf{f}^T (\mathbf{A}_n(\lambda) - \mathbf{I})^2 \mathbf{f} + \sigma^2 \mathrm{tr}[\mathbf{A}_n(\lambda)^2] \right) \\
&= \frac{1}{n} \sum_{j=1}^{n} \frac{\lambda^2 \mu_j^2 b_j^2}{(1+\lambda\mu_j)^2} + \frac{\sigma^2}{n} \sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2} \\
&\leq \frac{\lambda}{n} \sum_{j=1}^{n} \frac{\lambda\mu_j}{(1+\lambda\mu_j)^2} \mu_j b_j^2 + \frac{\sigma^2}{n} \sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2} \\
&\leq \frac{\lambda}{4n} \mathbf{f}^T \mathbf{M} \mathbf{f} + \frac{\sigma^2}{n} \sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2} \\
&= O(\lambda) + O(n^{-1} \lambda^{-\frac{d}{2m}}),
\end{aligned}$$

where $\mathbf{b} = \mathbf{U}^T \mathbf{f} = (b_1, b_2, ..., b_n)^T$, and $\mathbf{M} = \mathbf{U} \Lambda \mathbf{U}^T$.

Proof of Lemma 1.3.15

Proof Let $0 = \mu_1 < \mu_2 \leq \cdots \leq \mu_n$ be the eigenvalues of penalty matrix $\mathbf{M}$, and u_j

the unit eigenvector corresponding to μ_j, $j = 1, 2, \cdots, n$. So we have

$$\begin{aligned} n\mathbb{E}[r_n(\lambda)] &= n\mathbb{E}[n^{-1}\|\hat{\mathbf{f}}_n(\lambda) - \mathbf{f}\|^{\mathbf{2}}] \\ &= E[(\hat{\mathbf{f}}_n(\lambda) - \mathbf{f})^{\mathbf{T}}(\hat{\mathbf{f}}_{\mathbf{n}}(\lambda) - \mathbf{f})] \\ &= \mathbf{f}^{\mathbf{T}}(\mathbf{A}_{\mathbf{n}}(\lambda) - \mathbf{I})^{\mathbf{2}}\mathbf{f} + \sigma^{\mathbf{2}}\mathrm{tr}[\mathbf{A}_{\mathbf{n}}(\lambda)^{\mathbf{2}}] \\ &= \sum_{j=2}^{n} \frac{\lambda^2\mu_j^2}{(1+\lambda\mu_j)^2} b_j^2 + \sigma^2 \sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2}, \end{aligned}$$

where $b_j = \mathbf{u_j^T f}$.

If $\lambda \sim O(1)$ or $\lambda \to \infty$, since $\mu_j \sim j^{2m/d}$ for $j > 1$, there exists j^* such that $j^*/n \to 0$ and $\frac{\lambda\mu_j}{1+\lambda\mu_j} \geq \frac{1}{2}$ for $j > j^*$, then

$$\begin{aligned} n\mathbb{E}[r_n(\lambda)] &\geq \sum_{j=2}^{n} \frac{\lambda^2\mu_j^2}{(1+\lambda\mu_j)^2} b_j^2 \geq \frac{1}{4} \sum_{j>j^*} b_j^2 \\ &\geq \frac{n}{4}|f|_{T,0}^2 - \frac{1}{4} j^* \max\{b_1^2, \cdots, b_{j^*}^2\} = O(n) \to \infty. \end{aligned}$$

On the other hand, if $\lambda \to 0$ as $n \to \infty$, we have

$$n\mathbb{E}[r_n(\lambda)] = \sigma^2 \sum_{j=1}^{n} \frac{1}{(1+\lambda\mu_j)^2} = O(\lambda^{-\frac{d}{2m}}) \to \infty,$$

where the second equation is based on Lemma 1.3.13.

Proof of Lemma 1.3.16

Proof Due to the following decomposition of risk function:

$$\begin{aligned} &|\mathbb{E}[r_n(\lambda)] - r_n(\lambda)| \\ &= n^{-1} \left| \mathbf{f}^T(\mathbf{A}_n(\lambda) - \mathbf{I})^2\mathbf{f} + \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2] - \|(\mathbf{A}_n(\lambda) - \mathbf{I})\mathbf{f} + \mathbf{A}_n(\lambda)\varepsilon\| \right| \\ &= \left| \|\mathbf{A}_n(\lambda)\varepsilon\|^2 - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2] + \mathbf{f}^T \left(\mathbf{A}_n(\lambda) - \mathbf{I}_n\right) \mathbf{A}_n(\lambda)\varepsilon \right|. \end{aligned}$$

it suffices to show

$$\sup_{\lambda>0} \frac{n^{-1}|\mathbf{f}^T \left(\mathbf{A}_n(\lambda) - \mathbf{I}_n\right) \mathbf{A}_n(\lambda)\varepsilon|}{\mathbb{E}[r_n(\lambda)]} \xrightarrow{p} 0, \tag{30}$$

and

$$\sup_{\lambda>0} \frac{n^{-1}\left|\|\mathbf{A}_n(\lambda)\varepsilon\|^2 - \sigma^2 \mathrm{tr}[\mathbf{A}_n(\lambda)^2]\right|}{\mathbb{E}[r_n(\lambda)]} \xrightarrow{p} 0. \tag{31}$$

According to *Chebyshev Inequality*, for any given $\delta > 0$,

$$\begin{aligned}
&\mathbb{P}\left(\frac{n^{-1}|\mathbf{f}^T\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{A}_n(\lambda)\varepsilon|}{\mathbb{E}[r_n(\lambda)]} > \delta\right) \\
&\leq \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\mathbb{E}[(\mathbf{f}^T\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{A}_n(\lambda)\varepsilon)^2] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{f}\mathbf{f}^T\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{A}_n(\lambda)] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\sigma^2\|\mathbf{A}_n(\lambda)\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{f}\|^2 \\
&\leq \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-1}\sigma^2\frac{\|\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{f}\|^2}{n\mathbb{E}[r_n(\lambda)]} \\
&\leq \sigma^2(n\mathbb{E}[r_n(\lambda)])^{-1} \to 0,
\end{aligned}$$

since $n\mathbb{E}[r_n(\lambda)] \geq \|\left(\mathbf{A}_n(\lambda)-\mathbf{I}_n\right)\mathbf{f}\|^2$. Thus (30) holds.

Again for any given $\delta > 0$, we have

$$\begin{aligned}
&\mathbb{P}\left(\frac{n^{-1}\left|\|\mathbf{A}_n(\lambda)\varepsilon\|^2 - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]\right|}{\mathbb{E}[r_n(\lambda)]} > \delta\right) \\
&\leq \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\mathbb{E}[\left(\|\mathbf{A}_n(\lambda)\varepsilon\|^2 - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]\right)^2] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-1}\frac{E[\|\mathbf{A}_n(\lambda)\varepsilon\|^4] - (\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2])^2}{n\mathbb{E}[r_n(\lambda)]}.
\end{aligned}$$

The last equality comes from the fact that $\mathbb{E}(\|\mathbf{A}_n(\lambda)\varepsilon\|^2) = \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]$ and

$$\mathbb{E}[\left(\|\mathbf{A}_n(\lambda)\varepsilon\|^2 - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]\right)^2] = \mathrm{Var}(\|\mathbf{A}_n(\lambda)\varepsilon\|^2).$$

We know from the beginning that $n\mathbb{E}[r_n(\lambda)] \geq \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]$, it suffices to show that

$$\frac{E[\|\mathbf{A}_n(\lambda)\varepsilon\|^4] - (\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2])^2}{\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2]} < \text{Constant}. \tag{32}$$

Denote $\mathbf{B} = \mathbf{A}_n(\lambda)^2 = (B_{ij})_{n\times n}$,then we have

$$\begin{aligned}
\mathbb{E}[\|\mathbf{A}_n(\lambda)\varepsilon\|^4] &= \mathbb{E}[(\varepsilon^T\mathbf{B}\varepsilon)^2] \\
&= \mathbb{E}[(\sum_{i,j} B_{ij}\varepsilon_i\varepsilon_j)(\sum_{k,l} B_{kl}\varepsilon_k\varepsilon_l)] \\
&\leq \left(\sum_{i=1}^n B_{ii}\sigma^2\right)^2 + \sum_{i=1}^n B_{ii}^2 E[\varepsilon_i^4] + \sum_{i\neq j} B_{ij}^2\sigma^4.
\end{aligned}$$

There exists a constant c such that $\mathbb{E}[\varepsilon_i^4] \leq c\sigma^2$ and $\sigma^4 \leq c\sigma^2$, so we get

$$\begin{aligned}
\mathbb{E}[\|\mathbf{A}_n(\lambda)\varepsilon\|^4] &= \left(\sum_{i=1}^n B_{ii}\sigma^2\right)^2 + c\sum_{i=1}^n B_{ii}^2\sigma^2 + c\sum_{i\neq j} B_{ij}^2\sigma^2 \\
&= \left(\sum_{i=1}^n B_{ii}\sigma^2\right)^2 + c\sum_{i,j} B_{ij}^2\sigma^2 \\
&= (\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2])^2 + c\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^4] \\
&\leq (\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2])^2 + c\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2],
\end{aligned}$$

which implies (32) and completes the proof.

Proof of Lemma1.3.17

Proof From Section. 1.3.2 we know $\mu_i = O(i^{\frac{2m}{d}})$. Then

$$\begin{aligned}
\lim_{n\to\infty} \frac{\left(n^{-1}\sum_{i=l+1}^n k_i^{-1}\right)^2}{n^{-1}\sum_{i=l+1}^n k_i^{-2}} &= \lim_{n\to\infty} \frac{\left(\sum_{i=l+1}^n \mu_i^{-1}\right)^2}{n\sum_{i=l+1}^n \mu_i^{-2}} \\
&= \lim_{n\to\infty} \frac{\left(\int_{l+1}^n \mu^{-2m/d}d\mu\right)^2}{n\int_{l+1}^n \mu^{-4m/d}d\mu} \\
&= \lim_{n\to\infty} \frac{(4m-d)d}{(2m-d)^2} \cdot \frac{\left((l+1)^{1-\frac{2m}{d}} - n^{1-\frac{2m}{d}}\right)^2}{n\left((l+1)^{1-\frac{4m}{d}} - n^{1-\frac{4m}{d}}\right)} \\
&= \lim_{n\to\infty} \frac{(4m-d)d}{(2m-d)^2} \cdot \frac{l+1}{n} \cdot \frac{\left(1-(\frac{l+1}{n})^{\frac{2m}{d}-1}\right)^2}{1-(\frac{l+1}{n})^{\frac{4m}{d}-1}} \\
&= 0.
\end{aligned}$$

Proof of Lemma 1.3.18

Proof From the fact that

$$\sigma^2(n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)])^2 \leq \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)^2] \leq \mathbb{E}[r_n(\lambda_n)] \to 0,$$

therefore $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)] \to 0$, and thus

$$n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)] \to 1.$$

By the fact that $n^{-1}\|\varepsilon\|^2 \to \sigma^2$ and Cauchy-Schwartz inequality,

$$n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2 = n^{-1}\|\varepsilon\|^2 + n^{-1}\|\mathbf{f} - \hat{\mathbf{f}}_n(\lambda_n)\|^2 + \frac{2}{n}\left|(\mathbf{f} - \hat{\mathbf{f}}_n(\lambda_n))^T\varepsilon\right| \to \sigma^2.$$

Proof of Lemma 1.3.19

Proof Recall $\mathbf{A}_n(\lambda_n) = (\mathbf{I}_n + \lambda_n\mathbf{M})^{-1} = (\mathbf{I}_n + \mathbf{K}_n(\lambda_n))^{-1}$. Obviously

$$\frac{(n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)])^2}{n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)^2]} = \frac{(n^{-1}\sum_{i=1}^n(1+\kappa_i)^{-1})^2}{n^{-1}\sum_{i=1}^n(1+\kappa_i)^{-2}}, \tag{33}$$

where $0 \le \kappa_1 \le \cdots \le \kappa_n$ are the eigenvalues of $\mathbf{K}_n(\lambda_n)$. Let l be the number holding $\kappa_l \le 1 < \kappa_{l+1}$, then we have

$$\sum_{i=1}^n(1+\kappa_i)^{-1} \le l + \sum_{i=l+1}^n \kappa_i^{-1},$$

and

$$\sum_{i=1}^n(1+\kappa_i)^{-2} \ge \frac{1}{4}\left(l + \sum_{i=l+1}^n \kappa_i^{-2}\right).$$

Then it suffices to show

$$\left(\frac{l}{n} + \frac{1}{n}\sum_{i=l+1}^n \kappa_i^{-1}\right)^2 \Big/ \frac{1}{4}\left(\frac{l}{n} + \frac{1}{n}\sum_{i=l+1}^n \kappa_i^{-2}\right) \to 0.$$

$\mathbb{E}[r_n(\lambda_n)] \to 0$, since $r_n(\lambda_n)$ is nonnegative, thus we have $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)^2] \to 0$, from which we arrive at our result.

Proof of Lemma1.3.20

Proof We first prove (25), which can be rewritten as

$$\begin{aligned}
&\frac{2\left|\frac{\sigma^2\mathrm{tr}[\mathbf{I}_n-\mathbf{A}_n(\lambda)]\mathbf{y}^T(\mathbf{I}_n-\mathbf{A}_n(\lambda))\varepsilon}{n\|(\mathbf{I}_n-\mathbf{A}_n(\lambda))\mathbf{y}\|^2} - \frac{\sigma^4(\mathrm{tr}[\mathbf{I}_n-\mathbf{A}_n(\lambda)])^2}{n\|(\mathbf{I}_n-\mathbf{A}_n(\lambda))\mathbf{y}\|^2} - n^{-1}\|\varepsilon\|^2 + \sigma^2\right|}{r_n(\lambda)} \\
&\le 2\frac{\sigma^2\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2} \cdot \frac{n^{-1}\left|\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))\varepsilon\right|}{r_n(\lambda)} \\
&+ 2\frac{\sigma^2\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2} \cdot \frac{n^{-1}\left|\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)]\right|}{r_n(\lambda)} \\
&+ 2\frac{\left|\left(\frac{\sigma^2\mathrm{tr}[\mathbf{I}_n-\mathbf{A}_n(\lambda)]}{\|(\mathbf{I}_n-\mathbf{A}_n(\lambda))\mathbf{y}\|^2} - 1\right)(\sigma^2 - n^{-1}\|\varepsilon\|^2)\right|}{r_n(\lambda)}.
\end{aligned} \tag{34}$$

Note that $n^{-1}\text{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)] \to 1$, $n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2 \to \sigma^2$, from Lemma 1.3.18 and Lemma 1.3.16, it suffices to show the following three equations

$$\sup_{\lambda>0} \frac{n^{-1}\left|\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))\varepsilon\right|}{\mathbb{E}[r_n(\lambda)]} \to 0, \tag{35}$$

$$\sup_{\lambda>0} \frac{n^{-1}\left|\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\text{tr}[\mathbf{A}_n(\lambda)]\right|}{\mathbb{E}[r_n(\lambda)]} \to 0, \tag{36}$$

$$\sup_{\lambda>0} \frac{|(\sigma^2 n^{-1}\text{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda)] - n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2)(\sigma^2 - n^{-1}\|\varepsilon\|^2)|}{\mathbb{E}[r_n(\lambda)]} \to 0. \tag{37}$$

For (35), according to Chebyshev Inequality, for any given $\delta > 0$, we have

$$\begin{aligned}
\mathbb{P}\left(\frac{n^{-1}\left|\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda)\varepsilon)\right|}{\mathbb{E}[r_n(\lambda)]} > \delta\right) &\le \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\mathbb{E}[(\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))\varepsilon)^2] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\sigma^2\text{tr}[(\mathbf{I}_n - \mathbf{A}_n(\lambda))^T\mathbf{f}\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\sigma^2\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))^T\mathbf{f}\|^2 \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-1}\sigma^2\frac{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))^T\mathbf{f}\|^2}{n\mathbb{E}[r_n(\lambda)]} \\
&\le \delta^2\sigma^2(n\mathbb{E}[r_n(\lambda)])^{-1} \to 0.
\end{aligned}$$

For (36), again using Chebyshev inequality, for any given $\delta > 0$, we have

$$\begin{aligned}
&\mathbb{P}\left(\frac{n^{-1}\left|\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\text{tr}[\mathbf{A}_n(\lambda)]\right|}{\mathbb{E}[r_n(\lambda)]} > \delta\right) \\
&\le \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-2}\mathbb{E}[(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\text{tr}[\mathbf{A}_n(\lambda)])^2] \\
&= \delta^{-2}(n\mathbb{E}[r_n(\lambda)])^{-1}\frac{\mathbb{E}[(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon)^2] - (\sigma^2\text{tr}[\mathbf{A}_n(\lambda)])^2}{n\mathbb{E}[r_n(\lambda)]}.
\end{aligned}$$

Since $n\mathbb{E}[r_n(\lambda)] \ge \sigma^2\text{tr}[\mathbf{A}_n(\lambda)^2]$, we only need to show the following

$$\frac{\mathbb{E}[(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon)^2] - (\sigma^2\text{tr}[\mathbf{A}_n(\lambda)])^2}{\sigma^2\text{tr}[\mathbf{A}_n(\lambda)^2]} < \text{Constant}. \tag{38}$$

Denote $\mathbf{A}_n(\lambda) = (A_{ij})_{n\times n}$, then we have

$$\begin{aligned}\mathbb{E}[(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon)^2] &= \mathbb{E}[(\sum_{i,j} A_{ij}\varepsilon_i\varepsilon_j)(\sum_{k,l} A_{kl}\varepsilon_k\varepsilon_l)] \\ &= \mathbb{E}[(\sum_{i} A_{ii}\varepsilon_i^2)(\sum_{k} A_{kk}\varepsilon_k^2)] + \mathbb{E}[(\sum_{i\neq j} A_{ij}\varepsilon_i\varepsilon_j)(\sum_{k\neq l} A_{kl}\varepsilon_k\varepsilon_l)] \\ &\leq \left(\sum_{i=1}^n A_{ii}\sigma^2\right)^2 + \sum_{i=1}^n A_{ii}^2\mathbb{E}[\varepsilon_i^4] + \sum_{i\neq j} A_{ij}^2\sigma^4.\end{aligned}$$

There exists a constant c such that $\mathbb{E}[\varepsilon_i^4] \leq c\sigma^2$ and $\sigma^4 \leq c\sigma^2$, we have

$$\begin{aligned}\mathbb{E}[(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon)^2] &\leq \left(\sum_{i=1}^n A_{ii}\sigma^2\right)^2 + c\sum_{ij} A_{ij}\sigma^2 \\ &= (\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)])^2 + c\sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)^2],\end{aligned}$$

which finishes our proof of (36).

For (37), using the proof of (35), (36) and $\sigma^2(n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)])^2 \leq \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)^2] \leq \mathbb{E}[r_n(\lambda)]$, we only need to show

$$\sup_{\lambda>0} \frac{|\sigma^2 - n^{-1}\|\varepsilon\|^2|}{(\mathbb{E}[r_n(\lambda)])^{1/2}} \to 0, \tag{39}$$

since the fact that

$$\begin{aligned}&\left|\sigma^2 n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda)] - n^{1-}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\|^2\right| \\ &= \left|\sigma^2 - \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)] - n^{-1}\|\varepsilon + \mathbf{f} - \hat{\mathbf{f}}_n(\lambda)\|^2\right| \\ &= \left|\sigma^2 - \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)] - n^{-1}\|\varepsilon\|^2 - r_n(\lambda) - 2n^{-1}(\mathbf{f} - \hat{\mathbf{f}}_n(\lambda))^T\varepsilon\right| \\ &\leq \left|\sigma^2 - n^{-1}\|\varepsilon\|^2\right| + r_n(\lambda) + 2n^{-1}\left|\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda)\varepsilon\right| \\ &+ 2n^{-1}\left|\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)]\right| + \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)].\end{aligned}$$

By the Chebyshev's inequality, for any given $\delta > 0$, we have

$$\begin{aligned}
&\mathbb{P}\left(\frac{|\sigma^2 - n^{-1}\|\varepsilon\|^2|}{(\mathbb{E}[r_n(\lambda)])^{1/2}} > \delta\right) \\
&\leq \delta^2(\mathbb{E}[r_n(\lambda)])^{-1}\mathbb{E}[(\sigma^2 - n^{-1}\|\varepsilon\|^2)^2] \\
&= \delta^2(\mathbb{E}[r_n(\lambda)])^{-1}\left(n^{-2}\mathbb{E}[\|\varepsilon\|^4] - \sigma^4\right) \\
&\leq \delta^2(\mathbb{E}[r_n(\lambda)])^{-1}\left(n^{-2}(n^2\sigma^4 + n\mathbb{E}[\varepsilon_i^4]) - \sigma^4\right) \\
&= \delta^2(n\mathbb{E}[r_n(\lambda)])^{-1}\mathbb{E}[\varepsilon_i^4] \to 0.
\end{aligned}$$

Now it remains to prove (26), the numerator of which can be rearranged as

$$\begin{aligned}
&n^{-1}\|\tilde{\mathbf{f}}_n(\hat{\lambda})\|^2 \\
&= \left(\frac{\sigma^2 n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda)]}{n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2} - 1\right)^2 n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2 \\
&= \frac{((\sigma^2 - n^{-1}\|\varepsilon\|^2) - r_n(\lambda) - 2A_1 + 2A_2 + \sigma^2 n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)])^2}{n^{-1}\|(\mathbf{I}_n - \mathbf{A}_n(\lambda))\mathbf{y}\|^2},
\end{aligned}$$

where $A_1 = \left(n^{-1}\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))\varepsilon\right)^2$ and $A_2 = n^{-1}(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)])$.

It is not hard to see that

$$\frac{(\sigma^2 - n^{-1}\|\varepsilon\|^2)^2}{r_n(\lambda)} \to 0, \tag{40}$$

$$\frac{\left(n^{-1}\mathbf{f}^T(\mathbf{I}_n - \mathbf{A}_n(\lambda))\varepsilon\right)^2}{r_n(\lambda)} \to 0, \tag{41}$$

$$\frac{\left(n^{-1}(\varepsilon^T\mathbf{A}_n(\lambda)\varepsilon - \sigma^2\mathrm{tr}[\mathbf{A}_n(\lambda)])\right)^2}{r_n(\lambda)} \to 0, \tag{42}$$

$$\frac{(n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda)])^2}{r_n(\lambda)} \to 0. \tag{43}$$

based on the proofs of Lemma 1.3.19.

Proof of Theorem1.3.21

In order to prove Theorem 1.3.21, we need the following lemmas.

Lemma 1.5.1 *Under the condition (A.2), we have $\tilde{r}_n(\lambda) \to 0$ when λ_n is from (A.2)*

Proof To get $\tilde{r}_n(\lambda_n) \to 0$, it suffices to show that

$$\frac{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]} \to \sigma^2, \tag{44}$$

since the following derivation

$$\begin{aligned}
\tilde{r}_n(\lambda_n) &= n^{-1}\|\tilde{\mathbf{f}}_n(\lambda_n) - \mathbf{f}\|^2 \\
&= n^{-1}\|\varepsilon - \sigma^2 \frac{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2 \\
&= n^{-1}\|\varepsilon - \sigma^2 \frac{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}(\varepsilon + \mathbf{f} - \hat{\mathbf{f}}_n(\lambda_n))\mathbf{y}\|^2 \\
&\le n^{-1}\left(1 - \sigma^2 \frac{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}\right)^2 \|\varepsilon\|^2 \\
&+ 2n^{-1}\left|1 - \sigma^2 \frac{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}\right| \|\varepsilon\|\|\mathbf{f} - \hat{\mathbf{f}}_n(\lambda_n)\| \\
&+ n^{-1}\left(\sigma^2 \frac{\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)]}{\|(\mathbf{I}_n - \mathbf{A}_n(\lambda_n))\mathbf{y}\|^2}\right)^2 \|\mathbf{f} - \hat{\mathbf{f}}_n(\lambda_n)\|^2.
\end{aligned}$$

Obviously, (44) follows from Lemma 1.3.18.

Lemma 1.5.2 *Under the condition (A.2), we have* $\tilde{r}_n(\hat{\lambda}_G) \to 0$.

Proof From the uniform consistency of $\mathrm{SURE}_n(\lambda)$ together with the fact that $\hat{\lambda}_G$ minimizes $\mathrm{SURE}_n(\lambda)$, we have

$$\begin{aligned}
\tilde{r}_n(\hat{\lambda}_G) &= \mathrm{SURE}_n(\hat{\lambda}_G) + o_p(1) \\
&\le \mathrm{SURE}_n(\lambda_n) + o_p(1) \\
&= \tilde{r}_n(\lambda_n) + o_p(1) = o_p(1).
\end{aligned}$$

This is equivalent to say that $\tilde{r}_n(\hat{\lambda}_G) \to 0$.

Lemma 1.5.3 *Under the condition (A.2), we have* $\mathrm{GCV}_n(\hat{\lambda}_G) \to \sigma^2$.

Proof This is trivial from Lemma 1.5.2.

Lemma 1.5.4 *If* ε_i*'s are i.i.d* $N(0, \sigma^2)$*, for any* $\delta > 0$*, we have*

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{\|(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_G))\mathbf{y}\|^2}{\|(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_G))\mathbf{f}\|^2 + \sigma^2 \mathrm{tr}[(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_G))^2]} \le 1 - \delta\right) = 0. \tag{45}$$

Proof According to the proof of Lemma 5.2 in [51], the above lemma can be established directly.

Lemma 1.5.5 *For any sequence* $\{\lambda_n\}$ *such that* $\mathrm{GCV}_n(\lambda_n) \to \sigma^2$ *under the condition (A.3), we have* $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)] \to 0$.

Proof Using Lemma 1.5.4 and $\{\lambda_n\}$ such that $\mathrm{GCV}_n(\lambda_n) \to \sigma^2$, we have

$$\left(n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n)]\right)^2 \geq \left(n^{-1}\mathrm{tr}[(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n))^2]\right)(1 - o_p(1)). \tag{46}$$

With the fact that $\left(n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n)]\right)^2 \leq n^{-1}\mathrm{tr}[(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n))^2]$, we get

$$\frac{\left(n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n)]\right)^2}{n^{-1}\mathrm{tr}[(\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n))^2]} \to 1. \tag{47}$$

Recall that $\mathbf{A}_n(\hat{\lambda}) = (\mathbf{I}_n + \hat{\lambda}_n\mathbf{M})^{-1} = (\mathbf{I}_n + \mathbf{K}_n(\hat{\lambda}_n))^{-1}$ and $0 \leq \kappa_1 \leq \cdots \leq \kappa_n$ are the eigenvalues of $\mathbf{K}_n(\hat{\lambda}_n)$. It is clear that $\mathbf{I}_n - \mathbf{A}_n(\hat{\lambda}_n)$ have eigenvalues $\{\frac{\kappa_i}{1+\kappa_i}\}$. Similarly as in [51], let κ be the random variable taking values κ_i with probability n^{-1} for each $i \in \{1, 2, ..., n\}$. Then (47) means

$$\frac{\kappa(1+\kappa)^{-1}}{\mathbb{E}[\kappa(1+\kappa)^{-1}]} \to 1.$$

This implies that both $\kappa_{[pn]}(1+\kappa_{[pn]})^{-1}$ and $\kappa_{[qn]}(1+\kappa_{[qn]})^{-1}$ tend to $\mathbb{E}[\kappa(1+\kappa)^{-1}]$, we have $\mathbb{E}[\kappa(1+\kappa)^{-1}] \to 1$, from which $n^{-1}\mathrm{tr}[\mathbf{A}_n(\hat{\lambda}_n)] \to 0$ follows.

Lemma 1.5.6 *For sequence* $\{\lambda_n\}$ *such that* $\mathrm{GCV}_n(\lambda_n) \to \sigma^2$, $\hat{\mathbf{f}}_n(\lambda_n)$ *is consistent if and only if* $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)] \to 0$.

Proof If $\hat{\mathbf{f}}_n(\lambda_n)$ is consistent, $r_n(\lambda_n) \to 0$ and hence $n^{-1}\|\mathbf{y} - \hat{f}_n(\lambda_n)\|^2 \to \sigma^2$ since $n^{-1}\|\varepsilon\|^2 \to \sigma^2$. Then from the fact that $\mathrm{GCV}_n(\lambda_n) = \frac{n^{-1}\|\mathbf{y}-\hat{\mathbf{f}}_n(\lambda_n)\|^2}{(n^{-1}\mathrm{tr}[\mathbf{I}_n-\mathbf{A}_n(\lambda_n)])^2} \to \sigma^2$, we have $(n^{-1}\mathrm{tr}[\mathbf{I}_n - \mathbf{A}_n(\lambda_n)])^2 \to 1$ and thus $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)] \to 0$.

Conversely, if $n^{-1}\mathrm{tr}[\mathbf{A}_n(\lambda_n)] \to 0$, since $\mathrm{GCV}_n(\lambda_n) \to \sigma^2$, we have $n^{-1}\|\mathbf{y} - \hat{f}_n(\lambda_n)\|^2 \to \sigma^2$. Then with the fact that $n^{-1}\|\varepsilon\|^2 \to \sigma^2$, we have $r_n(\lambda_n) \to 0$, which implies that $\hat{\mathbf{f}}_n(\lambda_n)$ is consistent.

From Lemmas 1.5.3, 1.5.5 and 1.5.6, Theorem 1.3.21 is proved.

Proof of Theorem 1.3.22

Proof From the condition (A.2), for λ_n^* which is the minimizer of $r_n(\lambda)$, we have $r_n(\lambda_n^*) \to 0$. According to Lemma 1.3.19 we have

$$\frac{(n^{-1}\mathrm{tr}[\mathbf{A_n}(\lambda_\mathbf{n}^*)])^\mathbf{2}}{n^{-1}\mathrm{tr}[\mathbf{A_n}(\lambda_\mathbf{n}^*)^\mathbf{2}]} \to 0.$$

Hence from Lemma 1.3.20, we know $\mathrm{SURE}_n(\lambda_n^*) - n^{-1}\|\varepsilon_n\|^2 + \sigma^2 = r_n(\lambda_n^*)(1 + o_p(1))$.

On the other hand, from Theorem 1.3.21 this also holds for $\hat{\lambda} = \hat{\lambda_G}$. Therefore we have

$$\mathrm{SURE}_n(\hat{\lambda}_G) - n^{-1}\|\varepsilon_n\|^2 + \sigma^2 = r_n(\hat{\lambda}_G)(1 + o_p(1)).$$

Since $\mathrm{SURE}_n(\hat{\lambda}_G) \le \mathrm{SURE}_n(\lambda_n^*)$ and $r_n(\lambda_n^*) \le r_n(\hat{\lambda}_G)$, we have $r_n(\hat{\lambda}_G)/r_n(\lambda_n^*) \xrightarrow{p} 1$.

1.5.2 Agmon's Theorem

Let $A(x, D) = \sum_{|\alpha| \le m'} a_\alpha(x) D^\alpha$ be an elliptic operator of order m' in Ω, having continuous leading coefficients and bounded, measurable lower order coefficients. Let A be symmetric over $C_0^\infty(\Omega)$ in the sense that for all $\phi, \psi \in C_0^\infty(\Omega), (A\phi, \psi)_{0,\Omega} = (\phi, A\psi)_{0,\Omega}$. Suppose that there exists an unbounded self-adjoint transformation $\mathcal{A}$ on $L_2(\Omega)$, such that $C_0^\infty(\Omega) \subset D(\mathcal{A}) \subset H_{m'}(\Omega)$ and $\mathcal{A}u = Au, \forall u \in D(\mathcal{A})$. Let $n' = n$ if n is odd, $n' = n + 1$ if n is even. In case $m' \le n'$, suppose that there exists an odd positive integer k such that $k > n'/m'$, the coefficients of A are in $C^{(k-1)m'*}(\Omega)$ and $D(\mathcal{A}^k) \subset H_{km'}(\Omega)$.

Then the spectrum of $\mathcal{A}$ is discrete, and the eigenvalues of $\mathcal{A}$ have finite multiplicity. Let $\{\lambda_j\}$ be the sequence of eigenvalues of $\mathcal{A}$ counted according to multiplicity. For $\lambda > 0$, let $N_+(\lambda)$ be the number of nonnegative eigenvalues $\lambda_j \le \lambda$, then

$$N_+(\lambda) = c\lambda^{n/m'} + o(\lambda^{n/m'})$$

as $\lambda \to \infty$, where $c = (2\pi)^{-n} \int_\Omega w(x)dx$, and $w(x) = |\{\xi : 0 < A'(x, i\xi) < 1\}|$. Note that $A'(x, \xi) = \sum_{|\alpha|=m'} a_\alpha(x)\xi^\alpha$.

1.5.3 Neumann Boundary Condition

Consider the following physical problem: a planar object is surrounded by material capable of transferring heat at a prescribed rate $f(x, y)$; our objective is to find the equilibrium temperature inside the object. The corresponding PDE problem is as follow: let Ω be a closed region of the plane, find the function $\phi(x, y)$ such that

$$\begin{aligned} \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} &= 0 \quad \forall (x, y) \in \Omega \\ \frac{\partial \phi}{\partial \mathbf{n}} &= kf(x, y) \quad \forall (x, y) \in \partial\Omega \end{aligned}$$

where $\mathbf{n}$ is the normal direction to the boundary. Such a PDE bondary value problem is called *Neumann problem*. It is also obvious that the physical problem is ill-posed unless the total rate at which heat flows into the object is 0, which requires the condition

$$\int_{\partial\Omega} \frac{\partial \phi}{\partial \mathbf{n}} ds = \int_{\partial\Omega} f(x, y) ds = 0$$

CHAPTER II

BEAMLET-BASED GRAPH STRUCTURE FOR PATH PLANNING USING MULTISCALE INFORMATION

2.1 Introduction

A new beamlet-based graph structure is proposed for efficient path-planning within an environment full of obstacles. The main idea is to use multi-scale information, by utilizing techniques similar to those used in [18] for the purpose of statistical image processing. The theoretical analysis shows that the proposed multiscale version of the well-known A* algorithm based on this new graph structure has lower worst-case complexity than the standard A* algorithm applied to the nearest-neighbor graph. In numerical experiments, we found that the proposed multiscale structure significantly reduced the number of node expansions in several large-scale environment scenarios.

The main idea of the proposed multiscale graph structure can be summarized as follows. Consider a uniform n by n grid representing the world (or an image) assuming, without loss of generality, 4-nearest-neighbor connectivity. There are $O(n^2)$ vertices and $O(n^2)$ edges in the corresponding graph. In order to reduce the number of node expansions (the most time-consuming step in all graph search algorithms), the proposed graph structure first employs a recursive dyadic partitioning (to be defined later) to divide the environment into "blocks" of different sizes. The block sizes are determined by the relative importance of information within those blocks. The collection of all blocks of the same size defines the *information scale*. The preprocessing of information within each block is conducted via an innovative *Bottom-Up Fusion* algorithm, which "fuses" multiscale information from finer scales to coarser scales. Therefore, a properly designed search algorithm, defined on the preprocessed

"blocks," can significantly reduce the number of vertices in the graph, while only slightly increasing the number of edges. As a result, path searching can be sped-up significantly, when preprocessing is feasible.

There are four major ingredients in the proposed multiscale approach, briefly summarized below (a full description is given in Section 3.4). (a) A recursive dyadic partitioning that divides the entire grid world into hierarchically organized *d-squares*. These d-squares are of different size, depending on the scale. Only a subset of these d-squares is used during path-planning. The concept of *Path-Finding Reduced Recursive Dyadic Partition* (PFR-RDP) is introduced to describe the collection of d-squares used for path-planning. It can be shown that the PFR-RDP contains at most $O(\log n)$ d-squares. (b) For each d-square, all free boundary cells (i.e., vertices) are connected by edges, with the edge weights being equal to the lengths of the corresponding shortest paths. (c) A fusion algorithm, which efficiently computes the weights in (b) using the recursive relationship between the dyadic squares across different scales. A new graph (based on the PFR-RDP and the weights being computed in (c)) is thus obtained. We call this new graph the *beamlet graph*, owing to its similarity with the data structure introduced in [18] to encode efficiently all linear features in an image[1]. (d) Finally, the A* algorithm (or any other similar graph search algorithm) is run on the beamlet graph to identify the optimal path.

As it will be shown in the sequel, the proposed beamlet graph has $O(n)$ vertices and $O(n^2)$ edges. The worst-case complexity of running A* or Dijkstra's algorithm on the original 4-nearest-neighbor graph is $O(n^2 \log n)$, assuming a Fibonacci heap is used. The complexity on the newly designed beamlet graph is $O(n^2)$. The reduction

[1]Note that the beamlet graph defined in the current chapter is different than the beamlet graph introduced in [18]; we have still decided to use the term *beamlet graph*, because both take advantage of "long-distance" neighboring relations between the nodes of the graph. This slight abuse of terminology should not cause a confusion—the two names attach to different problems.

by a factor of $\log n$ initially may not seem impressive; however, our numerical simulation results proved to be much more encouraging. In some cases, our approach demonstrates an increase in speed of the A* algorithm by one or two orders of magnitude. The numerical experiments show that by combining the new beamlet-based graph structure with the A* algorithm yields faster query time, adequate usage of memory and reasonable preprocessing time.

The major contributions of our work are summarized below. First, we introduce a new, beamlet-based graph structure, which uses the multiscale information from the environment in order to reduce the complexity of benchmark path-finding algorithms. Although tested extensively against the standard A* and Dijkstra algorithms, the proposed data structure is not tied to a specific graph search algorithm, and it can be easily incorporated into other existing approaches, such as A* with stronger heuristics (i.e., true distance heuristic, differential heuristic, etc [22, 68]), bidirectional search algorithms, etc [31, 14, 59]. Second, we provide a systematic approach, the *bottom-up fusion* algorithm, that allows us to "fuse" local information from finer scales into global information at coarser scales. This algorithm is of more general interest, as it can be employed whenever a multi-scale partition of the environment is available.

The rest of this chapter is organized as follows. Section 3.2 gives a description of the problem formulation of the path-planning problem. Section 3.4 provides the details of the proposed beamlet-based graph structure, which contains the dyadic partition tree construction, and the bottom-up fusion information collection method. The proposed multiscale version of A* (m-A*) applies A* on the resulting beamlet graph. The order of complexity of the m-A* is analyzed in Section 2.4. Section 3.6 compares m-A* with the benchmark A* algorithm via numerical experiments under different scenarios. Since the performance of A* strongly depends on the heuristic used, we also compare m-A* and A* in terms of a much stronger heuristic than the standard L_1 distance, namely, the true distance heuristic (TDH) [22, 68]. As

shown in Section 2.5.2, m-A* outperforms A* even when a much stronger heuristic is used; furthermore, the performance gap increases with the size of the problem data. Section 4.5 offers a brief overview of the theory behind multiscale beamlet analysis, along with a comparison of our approach with related work from the path-planning literature. Section 4.7 summarizes the results of this chapter and provides some suggestions for possible future extensions.

2.2 Problem Formulation

A deterministic path-planning problem consists of a graph $G = (V, E)$, where V is the set of vertices (i.e., the possible vehicle locations) and E is the set of edges, representing transitions between these vertices. The weight of each edge represents the cost of transitioning between the two corresponding vertex (viz. node) locations. Planning a path from an initial vertex to an end vertex can be cast as a single-pair, shortest path problem on this graph. In the deterministic (respectively, dynamic) path-planning problem, the environment does not (respectively, does) change over time.

We consider path-planning problems in a deterministic 2-D environment. Without loss of generality, we assume that the information about the environment is given via an n by n square image, where n is dyadic: $n = 2^J$ and J is a positive integer. Note that such an image-based formulation is well-adopted in the path-planning literature. Often, the image is called the *gridworld* [23]. The proposed method does not require the image to be square. However, assuming a squared image simplifies our algorithmic description. Therefore, henceforth, we will assume squared images. We will also assume that the image contains two types of pixels: black pixels (representing non-traversable *obstacles*) and white pixels (representing traversable *free cells*). The path-planning problem is to find the shortest path between a given pair of *source* and *destination* pixels.

Two popular shortest-path search algorithms in a deterministic setting are Dijkstra's algorithm [15] and the A* algorithm [37]. Both algorithms give the optimal path, and can be considered as special implementations of dynamic programming [50]. A* operates essentially the same way as Dijkstra's algorithm, except for the fact that it uses a heuristic estimate to guide the search towards the most promising states. The use of heuristics in A* potentially reduces the computational time.

To apply either search algorithm, one needs to first construct the search graph. In this graph each free cell is defined to be a vertex (viz. node), and, correspondingly, it is connected only to its free four nearest-neighbors (*four-nearest-neighbor connectivity assumption*). However, both Dijkstra's and A* algorithms have the tendency to be slow as the space to be searched increases. Below we propose a *beamlet-based graph structure* that takes advantage of the sparse information induced by the quadtree decomposition in a hierarchy of dyadic squares, thus improving the performance of the standard A* algorithm when it is applied on the beamlet graph. As it will be shown in Section 2.4, such a strategy can reduce significantly the order of computational complexity, in the worst-case.

The main objective of the proposed new graph structure is to construct a smaller size graph on which the computational complexity of searching for the shortest path is efficiently reduced. The intuition for the ensuing problem size reduction is given as follows. The direct implementation of Dijkstra's or A* algorithm searches through all free cells in the environment. This can be overwhelmingly redundant: if, for instance, the origin and destination vertices are in the upper-left and bottom-right quadrants, respectively, it is not necessary to scan through all the free vertices in the upper-right and bottom-left quadrants. Instead, one only needs to consider the boundary white pixels of these two quadrants. An illustration of this idea can be seen in Fig. 1. Armed with this intuition, in the sequel we develop a new dynamic programming algorithm for path planning in a cluttered environment, which takes

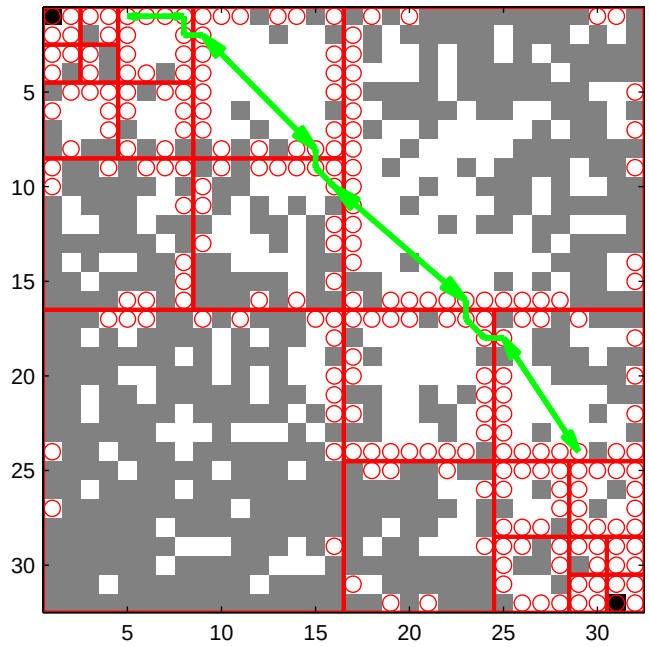


Figure 1: The free boundary cells attached to each dyadic square are the only ones that need to be considered in the proposed approach. Note that not all free boundary cells need to be expanded when running Dijkstra's or the A* algorithm on the beamlet graph. This figure also illustrates the vertices in the beamlet graph, defined in Section 2.3.2.

advantage of preprocessed information organized in a multiscale fashion, analogous to the quadratic tree structure that has been used in beamlet analysis [18]. The algorithm is explained in detail in the following section.

2.3 Multiscale Path Planning Strategy with Preprocessed Information

Each free cell is a vertex in the nearest neighbor graph and is connected to the free cells among its four nearest-neighbors. Each edge has unit weight. We describe our approach in four steps. In Section 2.3.1, we describe the *recursive dyadic partition* and the *path-finding reduced recursive dyadic partition*. These serve as the starting points of our approach. We then describe a new type of connectivity (Section 2.3.2), motivated by the beamlet structure introduced in [41, 16, 17]. We call the new data structure the *beamlet graph*, owing to its similarity with the connectivity relations

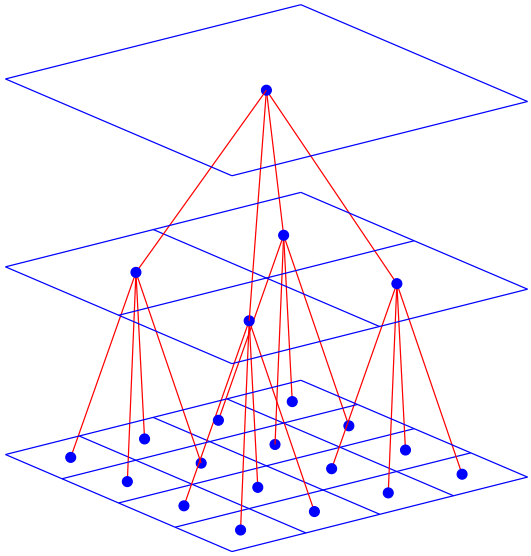
arising in the beamlet graph structure of [16]. To compute the edge weights needed to create the beamlet graph, a bottom-up fusion algorithm is given in Section 2.3.3. The proposed multiscale A* algorithm, essentially runs A* (or Dijkstra's) algorithm on the aforementioned beamlet graph. This is explained in detail in Section 2.3.4.

2.3.1 Recursive Dyadic Partitioning of the Environment

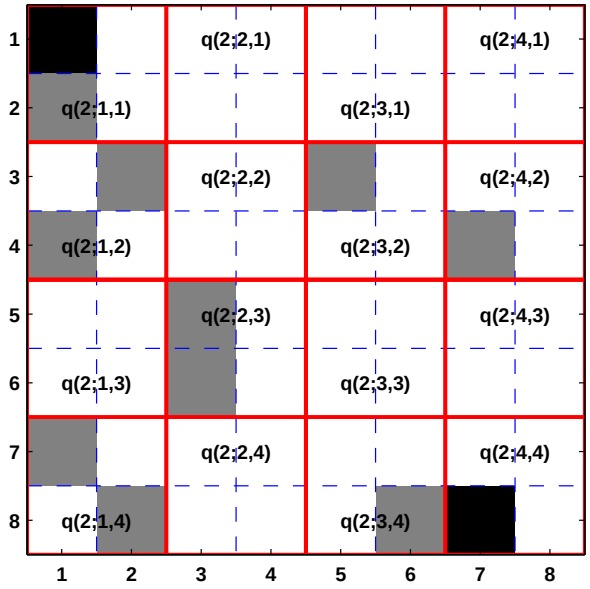
We describe two types of recursive dyadic partitioning (RDP). The first one is the complete version of RDP. We then introduce the *Path-Finding Reduced* RDP (PFR-RDP), which will play an important role in defining the beamlet graph—the graph structure that we will rely on.

The complete recursive dyadic partition can be described in a top-down approach: a squared image is subdivided into smaller d-squares, repeating the partitioning until the finest resolution of the image is reached. Let s ($1 \leq s \leq J$) denote the *scale*. A dyadic square (referred to as a *d-square* from this point on) at scale s, indexed by a, b ($1 \leq a, b \leq 2^s$) will be denoted by $q(s; a, b)$. We have $q(s; a, b) = \{(i, j) : 2^{J-s}(a-1)+1 \leq i \leq 2^{J-s}a, 2^{J-s}(b-1)+1 \leq j \leq 2^{J-s}b\}$. The d-square $q(s; a, b)$ at scale s can be partitioned into four d-squares at scale $s+1$; i.e., we have $q(s; a, b) = q(s+1; 2a-1, 2b-1) \cup q(s+1; 2a-1, 2b) \cup q(s+1; 2a, 2b-1) \cup q(s+1; 2a, 2b)$. Accordingly, we say that the d-square $q(s; a, b)$ has four children. The family of d-squares at all scales forms a quadtree. The correspondence between the recursive dyadic partitioning and the quadtree is illustrated in Figure 14.

In the path-planning problem, for a given pair of start and destination cells, only part of the complete RDP is needed. This is the Path-Finding Reduced RDP (PFR-RDP). The PFR-RDP is generated as follows. The image is initially subdivided into four equal, smaller d-squares. If either the source or the destination lies in a smaller d-square, the dyadic subdivision of this d-square will continue, unless the finest resolution has been reached. If a d-square contains neither the origin nor the



(a) A complete quadtree.



(b) The corresponding partition of a squared image.

Figure 2: (a) A complete recursive dyadic partition with the corresponding quadtree; (b) Complete recursive dyadic partition on a simple 8×8 image. The black cells are the source and destination and the gray cells are obstacles. The d-squares shown in the figure all come from the third (bottom) layer of partition in (a).

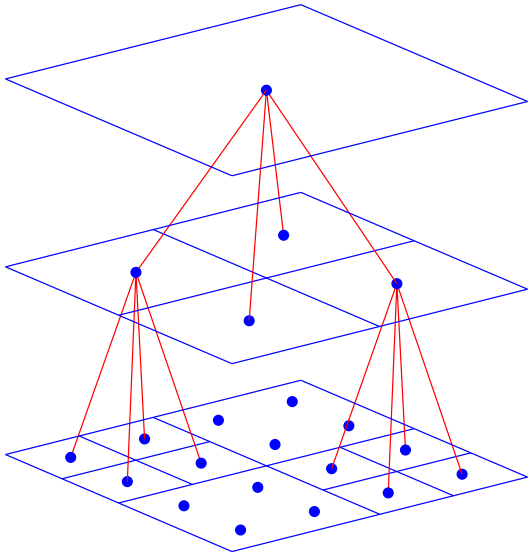
destination, no further partitioning is done to this d-square. The resulting partition is a partial recursive dyadic partition—not all d-squares are partitioned to the finest resolution—and corresponds to a partial quadtree. Figure 3 shows an example of a PFR-RDP along with the corresponding PFR-quadtree. The pseudo-code for the PFR-RDP is given in Algorithm 4.

Algorithm 1 PFR-RDP (Path-finding reduced recursive dyadic partitioning)

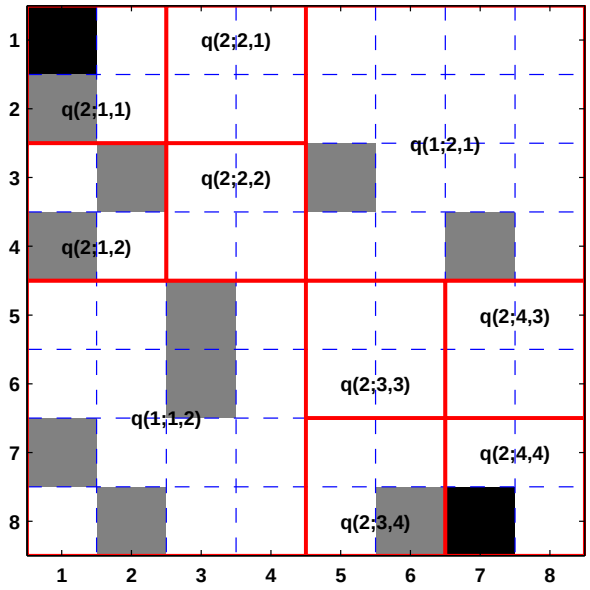
Set the largest scale to $J = \log_2 n$, where the image size is n by n.
Initialize the list *dptree* = [1, 1, 1]–the d-square at the coarsest level.
for $s = 1 : J - 1$ **do**
    For d-square at scale s in *dptree*
    if v_s (source) or v_e (destination) is in this d-square **then**
        In *dptree*, remove the line corresponding to this d-square;
        Partition into four equal, smaller d-squares, and insert them as new lines in *dptree*
    end if
end for
return *dptree*

2.3.2 Beamlet-like Connectivity

Recall that in the nearest neighbor graph, only the four nearest neighbors of a cell are connected by edges. Here we introduce another type of connectivity that takes advantage of connections between faraway cells. We will see that such a connectivity, together with the aforementioned PFR-RDP, can reduce the computational complexity of the search algorithm. For each fixed d-square, we only consider the free cells on its boundary. A pair of free cells on the boundary of the d-square are said to be connected by an edge if and only if there exists a feasible path between the two within this d-square. Note that such a definition is similar to the concept of beamlets introduced in [16]. We refer to Section 2.6.1 for the background theory on beamlet analysis. Representative original beamlets in [16] are displayed in Figure 4. Figure 5 shows several "beamlets" in the context of the current chapter attached to a



(a) A path-finding reduced quadtree.



(b) The corresponding partition of a squared image.

Figure 3: (a) A partial recursive dyadic partition and the corresponding PFR-RDP; (b) A partial recursive dyadic partition on a simple 8×8 image. The black cells denote the source and destination. The gray cells are obstacles. The d-squares shown in figure are from the third (bottom) layer in (a).

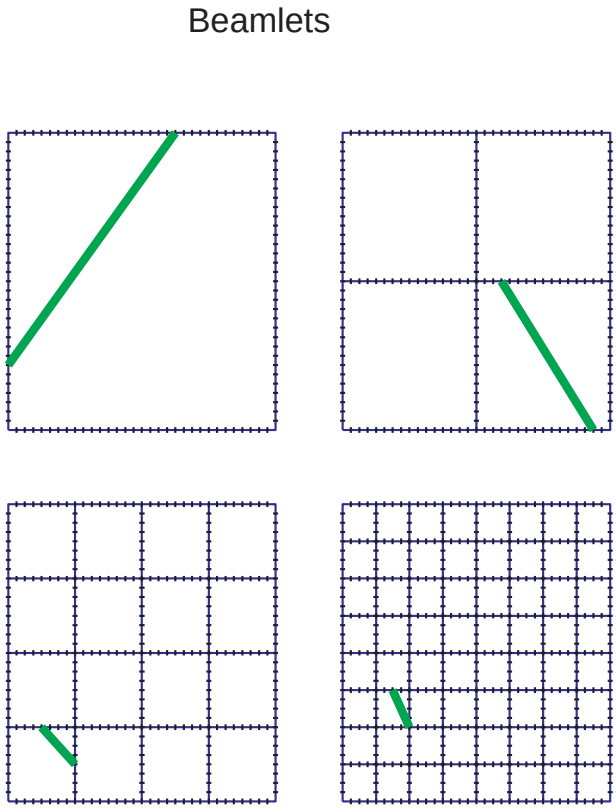
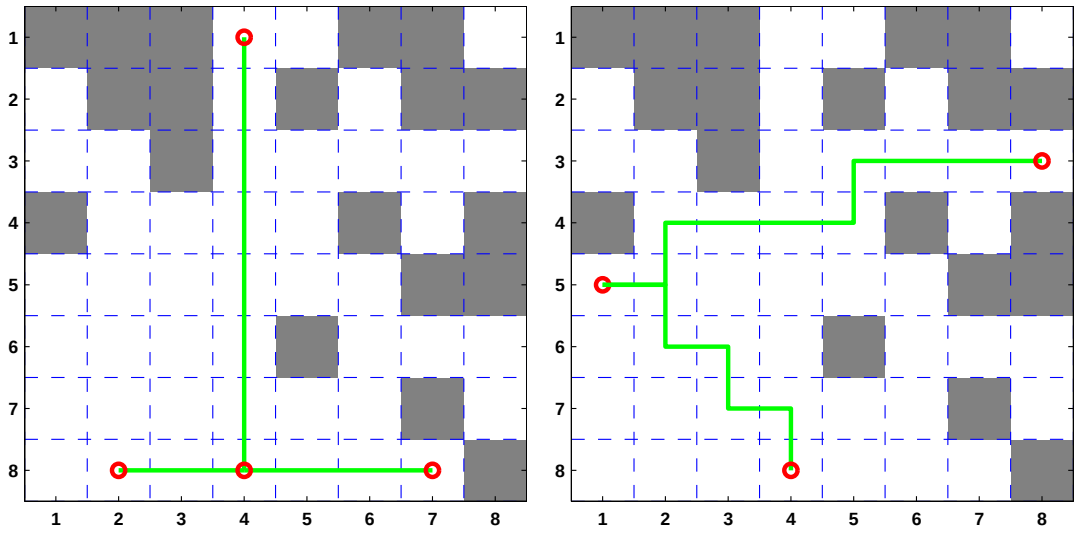


Figure 4: Illustrations of the original beamlets. The successive subdivision of the sides of the squares provides a hierarchical data structure that efficiently encodes the distance between any two points at the boundaries of the squares.

8×8 dyadic square. The green lines[2] show the corresponding shortest paths (i.e., the optimal beamlets) between two pairs of boundary free cells. Notice that a beamlet in the shortest-path problem may not be a straight line (Figure 5(b)). The beamlet



(a) Straight Beamlet. (b) Non-Straight Beamlet.

Figure 5: (a) Three straight beamlets in a 8×8 image; the red circles are the end points of each beamlet; (b) Two beamlets connecting the cells (1,5)-(4,8) and (1,5)-(8,3); note that beamlets in the shortest-path problem may not be straight lines.

graph is now defined as follows. First, the PFR-RDP of the given n by n image is obtained. All the free cells on the boundaries of each d-square in the PFR-RDP are defined to be the vertices in the beamlet graph. Two vertices in the beamlet graph are connected under two conditions: (a) they are nearest neighbors; or (b) they belong to the same d-square in the PFR-RDP and there is a beamlet (i.e., a feasible path inside the d-square) connecting them. Within each d-square, the weight of an edge is the length of the shortest path connecting the two vertices. An example of a PFR-RDP partition is shown in Figure 1. In Figure 1 the red grid shows the partial dyadic partition corresponding to the PFR-RDP. The red circles are the free boundary cells, i.e., the vertices in the beamlet graph.

To compute the weights of the edges within all d-squares in a PFR-RDP we make

[2]Please refer to the electronic version of the chapter for the color versions of the figures.

use of the following bottom-up fusion algorithm.

2.3.3 Bottom-Up Fusion Algorithm

The proposed multiscale path planning strategy requires the availability of the shortest path distances between any pair of free boundary cells for each d-square. When $s = J$ or $J - 1$, there are one or four pixels in the d-square, respectively. Hence, it is straightforward to compute these distances. For the general case, recall that a d-square $q(s; a, b)$ can be partitioned into four smaller d-squares at scale $s + 1$. If we already know the inter-distances between the free boundary cells within each of the smaller d-squares, and by considering the connectivity of the free boundary cells that belong to neighboring d-squares, we can treat all free boundary cells of the four d-squares at scale $s+1$ as vertices in a "fused" graph, and run Johnson's algorithm [44] to compute all shortest paths. The distances between free cells from neighboring d-squares can be computed directly. Since there are no more than $n2^{2-s}$ of these cells, the search algorithm can be run efficiently. Figure 6 shows how this "fusion" of shortest distances is conducted recursively within an 8×8 d-square. During the first step, the inter-distances between the free boundary cells of the four d-squares on the finer layer are computed using Johnson's algorithm. The green arrows show some of these distances. The dashed lines in the second layer, corresponding to the solid grid partition, show the fusion of the inter-distances. The dashed lines at the center of Figure 6 indicate the final step of the fusion algorithm which yields the inter-distances between the pair of black cells. The pseudo-code for this bottom-up fusion algorithm is provided in Algorithm. 5.

Overall, we have a bottom-up fusion algorithm that computes the inter-distances between the free boundary cells of all d-squares. In other words, we have found the shortest paths between all boundary free cells. These are exactly the edge weights in the beamlet graph.

Algorithm 2 BottomUpFusion (For each d-square)

Read the parameters of each d-square: s (scale), a, b (location);
if $s = \log n - 1$ **then**
    Compute the free boundary cells as vertices (Trivial case: only four cells in the d-square)
    Calculate the four nearest neighbor connectivity (edges) within each d-square
    Run Johnson's algorithm on the resulting graph to obtain all pairs of shortest paths: *cgraph* and *pathList*.
end if
if $s > 1$ **then**
    [*graph1, path1*] = BottomUpFusion$(s+1, 2a-1, 2b-1)$
    [*graph2, path2*] = BottomUpFusion$(s+1, 2a, 2b-1)$
    [*graph3, path3*] = BottomUpFusion$(s+1, 2a-1, 2b)$
    [*graph4, path4*] = BottomUpFusion$(s+1, 2a, 2b)$
    Merge *graph1, ..., graph4* into *Graph* by adding the connected edges between neighboring d-squares
    Run Johnson's algorithm on *Graph* and get *cgraph* and *tmpPathList*
    Insert the missing parts of paths in *tmpPathList* from *path1,..., path4* to obtain *pathList*
    return *cgraph, pathlist* (i.e., the beamlet graph)
end if

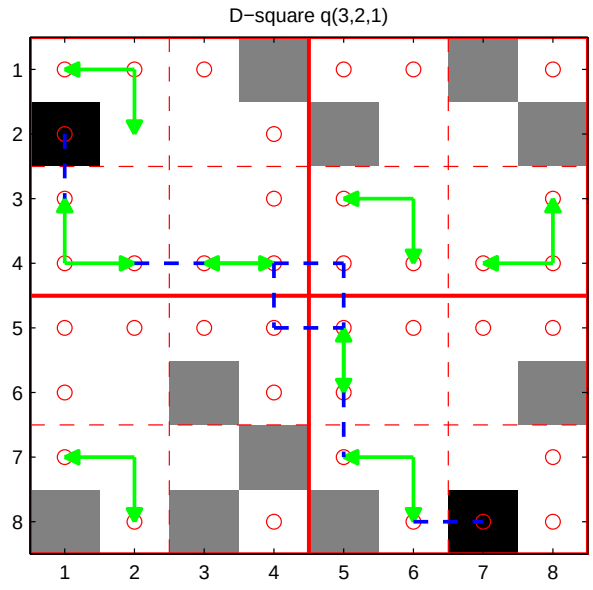


Figure 6: Bottom-up fusion in the d-square q(3,2,1) of Figure 1. The information fusion is conducted in a complete dyadic partition. Notice that the solid red grid stands for the partition corresponding to the second layer of the associated quadtree, and the dashed red grid corresponds to the partition with respect to the third layer. The green arrows indicate the inter-distances between free boundary cells. The blue lines show the fusion process.

2.3.4 Multiscale A* Algorithm on the Beamlet Graph

In the previous section, we described how we can obtain a beamlet-based graph structure. From this point on, we denote the beamlet graph as $BG = (V, E)$, where V denotes the vertices, i.e., the free boundary cells of all the d-squares in the PFR-RDP, and E denotes the edges representing the shortest distance paths between pairs of free boundary cells.

By using this new graph structure, we implement a multiscale version of the A* algorithm, henceforth called m-A* for short. The main steps of m-A* mimic the standard A* algorithm. For convenience, we repeat the main steps of the algorithm below. m-A* plans a path from the source vertex $v_s \in V$ to the destination vertex $v_e \in V$. To do this, the algorithm stores an estimate $g(v)$ of the path length from v_s to each vertex v. The algorithm also keeps an estimate of the path length from v to the destination denoted by $f(v) = g(v) + h(v, v_e)$. Initially, we set $g(v) = \infty$ for all vertices in V. The algorithm begins by setting $g(v_s) = 0$ and then places this vertex in a priority queue, known as the *OPEN* list. Each element v in this queue is ordered according to its f-value, that is, the sum of its current path length from v_s, stored in $g(v)$, and a heuristic estimate of the path length to the destination, $h(v, v_e)$. The vertex having the minimum f-value is pushed to the front of the priority queue. To be admissible, the heuristic $h(v, v_e)$ should underestimate the cost of the optimal path from v to v_e [38]. In our implementation, the heuristic estimate used was the usual L_1 distance. It should be emphasized, however, that this choice can be made without loss of generality. Other stronger heuristics and/or the use of a bi-directional search could have been used, with the results essentially remaining the same (see Section 2.5.2).

The algorithm pops the vertex v at the front of the queue and updates the g-values of all vertices in V representing obstacle-free cells that are reachable from this vertex through a direct edge (denoted as $Succ(v)$): if the value of $g(v)$ plus the weight of the

edge between v and its neighboring cell v' in the beamlet graph (denoted as $c(v, v')$) is less than the current g-value of the vertex v', then the g-value of v' is set to this new, lower value. If the g-value of a neighboring vertex v' changes, it is placed in the *OPEN* list. The algorithm continues exploring all vertices in the queue until it arrives at the destination vertex. At this stage, if the heuristic is admissible, then the path length from v_s to v_e is guaranteed to be optimal. The complete algorithm is given in Algorithm 6.

2.4 Complexity Analysis

In this section we provide a detailed analysis for the computational complexity of m-A*. First, we discuss the bottom-up fusion part, and then move on to the overall complexity of m-A*.

2.4.1 Complexity of Information Fusion Part

We derive an upper bound for the algorithmic complexity of the bottom-up fusion algorithm, i.e., we consider the worst case. Let $T(m)$ denote the amount of computations required to find the inter-distances between all pairs of free boundary cells within an $m \times m$ d-square. A recursive relationship dividing a cell of dimensions $2m \times 2m$ to four equal $m \times m$ squares can be written as follows,

$$T(2m) = 4T(m) + f(2m), \tag{48}$$

where $f(2m)$ denotes the effort for solving the all-pair shortest path problem during the information fusion step. Figure 7 shows an example of the fusion step within a d-square under the worst case scenario (no obstacles). Both connections of inter-distances between the same d-square (represented in (48) by $T(m)$), and between neighboring d-squares (represented in (48) by $f(2m)$) are shown in this figure.

A key step during the fusion process is the solution of the all-pair shortest path problem on the given graph. For the graph shown in Figure 7, for instance, one needs

Algorithm 3 Multiscale A*

Initialize *img,* v_s, v_e
Conduct *PFR-RDP* and obtain *dptree*
Run the *Bottom-Up Fusion* algorithm on each d-square in *dptree* and get beamlet graph
for all $v \in V$ **do**
    $g(v) = \infty$
end for
$h(v_s, v_e) = heuristicEstimate(v_s, v_e)$
$g(v_s) = 0$; $f(v_s) = h(v_s, v_e)$
$CLOSE := \emptyset$
$OPEN :=$ v_s with value $g(v_s) + h(v_s, v_e)$
while $OPEN \neq \emptyset$ **do**
    $v = \text{argmin}_{u \in OPEN}(g(u) + h(u, v_e))$
    remove state v from *OPEN*, add it into *CLOSE*
    if $v = v_e$ **then**
        return construct optimal path
    end if
    for all $v' \in Succ(v)$ **do**
        if $v' \in CLOSE$ **then**
            continue
        end if
        if $v' \in OPEN$ **then**
            if $g(v') > g(v) + c(v, v')$ **then**
                $g(v') = g(v) + g(v, v')$
                update v' with value $g(v') + h(v', v_e)$
            end if
        else
            insert v' into *OPEN* with value $f(v') = g(v') + h(v', v_e)$
        end if
    end for
end while
return failure

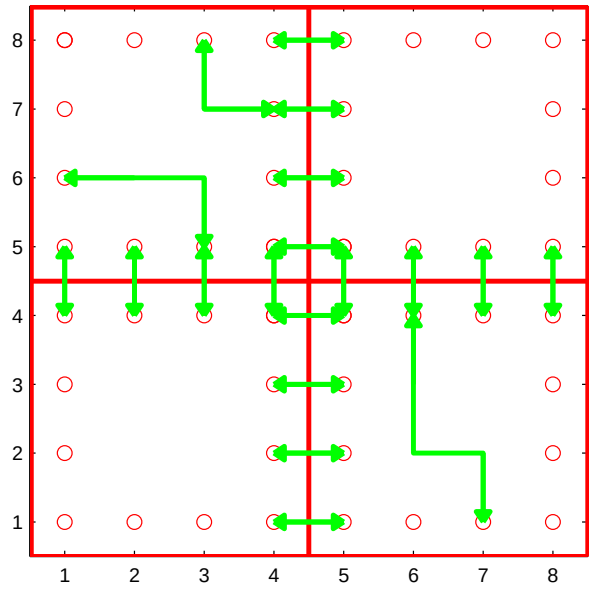


Figure 7: Illustration for the complexity analysis of the fusion algorithm. The three right-angle arrows provide examples of inter-distances between free boundary cells within two smaller scale d-squares. Along the common boundary of neighboring smaller scale d-squares, the straight arrow lines show the fusion of the inter-distances among these four d-squares. Note that for the purposes of complexity analysis, this figure illustrates the worst case scenario (i.e., no obstacles exist and hence all boundary cells at both levels are free).

to consider all boundary pixels of the smaller d-squares as vertices in the beamlet graph. Also note that there are at most $4(m-1)$ boundary pixels per smaller d-square. It follows that $|V| \leq 4 \times 4(m-1)$. The edges in the beamlet graph belong into two categories:

(a) Edges connecting the free boundary cells within each smaller d-square.

(b) Edges connecting the nearest-neighbor pixels between neighboring d-squares.

The green arrows in Figure 7 show some of these edges. In each d-square, there are at most $\binom{4m-4}{2}$ edges; there are at most $4m$ edges connecting free cells that are nearest neighbors not in the same d-square. Hence, we have $|E| \leq 4\binom{4m-4}{2} + 4m < 32m^2$.

Lemma 2.4.1 *For the bottom-up fusion algorithm, we have $f(m) = O(m^3)$, where $f(m)$ as in* (48)*.*

Proof Johnson's Algorithm [44] is the standard algorithm for solving the all-pair shortest path problem. If one assumes that an intermediate step of Johnson's algorithm (an implementation of the Dijkstra's algorithm) is done via Fibonacci heap, then the overall complexity is $O(|V|^2 \log(|V|) + |V||E|) = O(m^2 \log(m) + m^3) = O(m^3)$. This is exactly the complexity of $f(m)$.

To evaluate $T(m)$, we will need the following Master Theorem [11, Sect. 4.3].

Theorem 2.4.2 (Master Theorem) *Consider the recurrent relation of an algorithm of the form*

$$T(m) = a\, T\Big(\frac{m}{b}\Big) + f(m), \quad a \geq 1,\, b > 1,$$

where m is the size of the entire problem, a is the number of subproblems in the recursion, m/b is the size of each subproblem (it is assumed that all subproblems are of the same size). Let $f(m)$ be the cost of the work done outside the recursive calls,

which includes the cost of dividing the problem, and the cost of merging the solutions to these subproblems.

Suppose the following two conditions are satisfied:

(a) $f(m) = O\left(m^{\log_b a+\epsilon}\right)$ *for some constant* $\epsilon > 0$*, and*

(b) $af\left(\frac{m}{b}\right) \leq cf(m)$ *for some constant* $c < 1$ *and sufficiently large* m*.*

Then $T(m) = O(f(m))$*.*

As a direct consequence of Theorem 2.4.2 we therefore have the following result.

Corollary 2.4.3 *For the* $T(m)$ *in* (48)*, we have* $T(m) = O(f(m)) = O(m^3)$*.*

Proof Take $a = 4$ and $b = 2$, and apply Theorem 2.4.2.

2.4.2 Complexity of Searching

We now consider the order of complexity for running A* (or Dijkstra's) algorithm on the beamlet graph (BG), and the 4-nearest-neighbor graph (NNG), respectively. We consider the beamlet graph first. In order to better illustrate the idea, consider the PFR-RDP and the corresponding beamlet graph for the scenario shown in Figure 8. Recall that the vertices in the beamlet graph are the free boundary cells in all the d-squares in the PFR-RDP. For an n by n image, there are two scale-1 d-squares, and six scale-s d-squares when $s \geq 2$. For a d-square at scale s, there are at most $n2^{2-s}$ free boundary pixels. Hence, the upper bound of the total number of vertices in the beamlet graph is: $2 \times 2n + 6 \times n + 6 \times \frac{n}{2} + 6 \times \frac{n}{4} + \cdots = 4n + 6n + 3n + \frac{3}{2}n + \cdots \leq 16n$. On the other hand, within a d-square at scale s, the number of the edges is at most $\binom{n2^{2-s}}{2}$. At scale s, the number of connected free cells belonging to different d-squares is at most $n2^{2-s}$. Hence, the upper bound of the number of edges is $2\binom{2n}{2} + 2n + 6\binom{n}{2} + 2 \times \frac{n}{2} + 6\binom{n/2}{2} + 2 \times \frac{n}{4} = 4n^2 + 2n + 3n^2 + n + \frac{3}{4}n^2 + ... \approx 8n^2$. Recall that the complexity of running Dijkstra's algorithm with Fibonacci heap is $O(|E| + |V| \log |V|)$ [11]. We therefore have the following theorem.

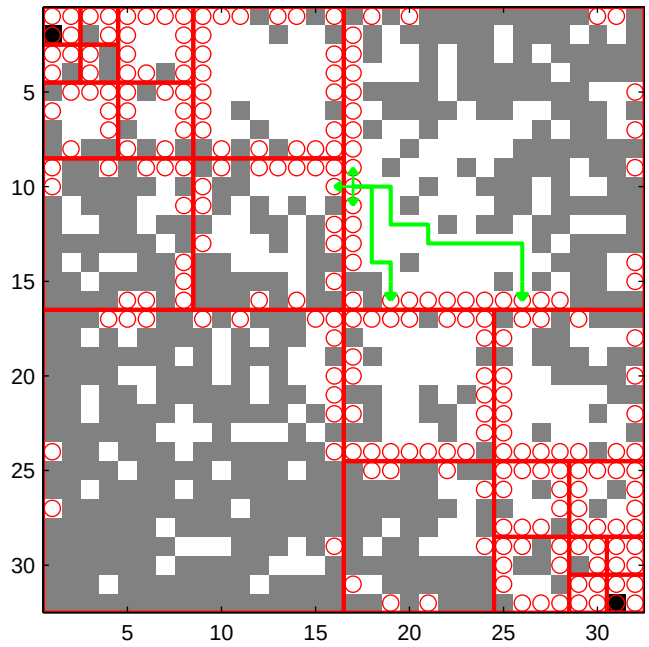


Figure 8: Illustration of the beamlet graph for a 32×32 image. The PFR-RDP is shown by the red grid lines. Free cells are marked by circles: these are the free boundary cells of the d-squares in the PFR-RDP. If two free cells belonging to different d-squares are nearest neighbors, they are also connected. All free cells within the same d-square are considered connected, as long as a feasible path exists.

Theorem 2.4.4 *The complexity of running Dijkstra's algorithm on the beamlet graph is* $O(n^2)$.

Proof The previous calculation gives the following estimates for the number of vertices and edges in the beamlet graph: $|V| = 16n$ and $|E| \approx 8n^2$, respectively. Using the known complexity bounds of Dijkstra's algorithm [15], the upper bound of the complexity is $O(n^2 + n \log n) = O(n^2)$.

Since A* and Dijkstra's algorithm have identical worst-case complexity [38], the following is immediate.

Corollary 2.4.5 *The complexity of running A* algorithm on the beamlet graph is* $O(n^2)$.

For comparison, in the nearest-neighbor graph we have $|V| = n^2$ and $|E| = 4n^2$. One can thus easily establish the following result.

Theorem 2.4.6 *If A* or the Dijkstra's algorithm is run on the nearest-neighbor graph, the worst-case complexity is* $O(4n^2 + n^2 \log n^2) = O(n^2 \log n)$.

The proof of the theorem is evident and is therefore skipped. Note, from Theorem 2.4.4 and Theorem 2.4.6, that running a search algorithm on the beamlet graph yields a reduction by a factor $\log n$ when compared to the same algorithm being run on the nearest neighbor graph.

2.4.3 Memory Usage

Since the beamlet-based graph structure involves preprocessing, the memory usage for storing the precomputed information needs to be estimated. Recall that the main step during processing is the bottom-up fusion algorithm, which is a recursive algorithm. In each recursion, only the free boundary cells from the four d-squares at the immediately finer scale are given as input to the all-pairs shortest path algorithm.

During the path-finding step, recall that in our complexity analysis we showed that, for an $n \times n$ image, the number of vertices in the beamlet graph has an upper bound of $16n$ and the number of edges has an upper bound of $8n^2$. Hence, the storage overhead for the beamlet graph is of order $O(n^2)$. Since we also need to store all shortest paths, the additional memory required is

$$\sum_{s=1}^{\log n} 6 \binom{n2^{2-s}}{2} 6\, n\, 2^{-s} \asymp O(n^3). \tag{49}$$

To see this, recall that for a scale s d-square, the number of edges is less than or equal to $\binom{n2^{2-s}}{2}$ and there are 6 (or less) d-squares at each scale s. It therefore requires no more than $6\, n\, 2^{-s}$ vertices to record the shortest path between any two free boundary cells within a scale s d-square. By summing over all scales, (49) results. In summary, the total storage overhead is, asymptotically, $O(n^3)$. The memory overhead for the nearest neighbor graph is easily computed to be $O(n^2)$, because in the nearest

neighbor graph both vertices and edges are of order $O(n^2)$ and one only needs to consider traversability between neighboring cells.

2.4.4 Preprocessing Time

The proposed multiscale strategy combines the bottom-up fusion algorithm of Section 2.3.3 with the search algorithm on the reduced-size beamlet graph. Recall that in each d-square of the PFR-RDP, an all-pair shortest path algorithm is solved. As seen already, the computational complexity of this preprocessing step is $O(n^3)$. Depending on the resolution used in the RDP, two extremes for the single-source path planning problem arise:

- Running a search algorithm (with Fibonacci Heap) on the entire environment (on the nearest neighbor graph), making no use of multiscale information. The overall complexity of this method is $O(n^2 \log n)$.

- Running an all-pairs shortest path finding algorithm (such as Johnson's algorithm) on the entire environment as the preprocessing step, which has complexity $O(n^4 \log n)$, and then extract the shortest distance for the given source and destination. The second step just involves a binary search and has complexity $O(\log n)$. The total complexity of this option is therefore $O(n^4 \log n)$.

The proposed m-A* algorithm falls in-between these two extremes (see also Table 1). In terms of applications, this flexibility can prove to be very useful. For instance, for many vehicles with embedded autonomous capabilities (e.g., ground robots, small UAVs) on-board processing of the available data can quickly become an implementation bottleneck during path-planning and execution. Making use of off-line (pre-computed) information is often helpful in practice to overcome this limitation. The proposed approach can be seen in this context as a compromise between off-line pre-processing and on-line search. Of course, in cases where the multiscale information of the environment is provided directly to the planner by a suitable sensor,

Table 1: Overall complexity comparison.

Complexity	No Preprocessing	m-A*	All-pairs Computed
Preprocessing	0	$O(n^3)$	$O(n^4 \log n)$
Query Time	$O(n^2 \log n)$	$O(n^2)$	$O(\log n)$

making use of the proposed multiscale graph structure will outperform traditional search algorithms by orders of magnitude, as demonstrated by both the previous complexity analysis and by our numerical studies.

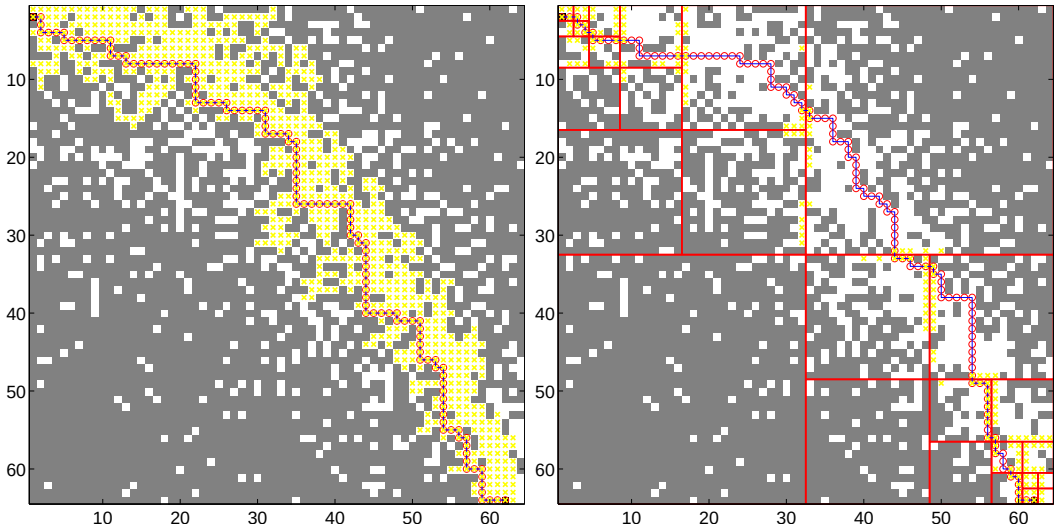
2.5 Numerical Studies

In this section, we provide numerical experiments to compare the performance from running the A* algorithm on the standard nearest neighbor graph and the proposed multiscale A* on the new beamlet graph. The comparison is based on the number of vertex expansions during the search. This is a more accurate criterion than, say, running time, since the query time heavily depends on the computer hardware used in each case. In these numerical experiments, the bottom-up fusion algorithm and the A* algorithm with Fibonacci heap were implemented in Matlab 2009(a) and C++, respectively, on an Intel Core2 Duo CPU 2.26 Ghz, with 1.89 GB of RAM, running Windows XP.

2.5.1 Comparison Based on L_1 Heuristic

In this section we provide the results from numerical simulations using three distinct cases of an obstacle-filled environment. In the first simulation, the probability of a certain cell (at location (x, y), where $1 \leq x, y \leq n$,) to be a free cell is assigned to be $p(x, y) = \exp(-\gamma|y - x^2/n|)$, where the constant γ will be specified later. The intuition of this model is that cells near the curve $y = x^2/n$ have higher probability to be free than cells far away from the same curve. This gridworld simulates the situation when there is a main "corridor" in the environment.

Table 2 contains the simulation results when $\gamma = 1/15$. The numbers in the first two columns indicate the number of vertex expansions in each case. The values in parentheses are the corresponding running times in milliseconds. The last column shows the ratios between the number of the expanded nodes used by the two algorithms. Figure 9 shows the comparison of the shortest path from the nearest-neighbor graph and the beamlet graph, respectively. The yellow crosses show the corresponding expanded vertices in both cases. As shown in this simulation, the m-A* algorithm clearly outperforms A* in terms of the number of vertex expansions.



(a) The shortest path and the expanded vertices in the nearest neighbor graph.

(b) The shortest path and the expanded vertices in the beamlet graph.

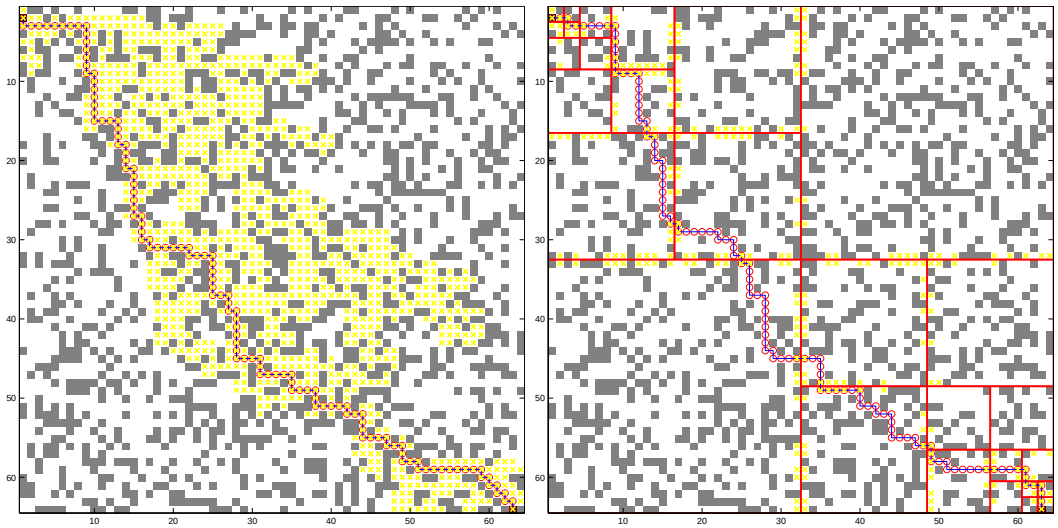
Figure 9: Example I: (a) The shortest path of NNG in a 64×64 image; the yellow crosses denote the expanded vertices during the search; (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.

The second simulation represents a much more difficult situation for path planning, and it involves a heavily cluttered environment (Figure 10). The presence of a large number of randomly distributed small obstacles will typically require a high number of vertex expansions for all search algorithms applied on the nearest neighbor graph.

A comparison of the vertex expansions (query time) for both A* and m-A* is given in Table 3. The table shows again the advantage of m-A* over the traditional A* algorithm in terms of the number of vertex expansions.

Table 2: Multiscale A* Algorithm Compared to Traditional A*. Example I.

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	644 (19)	152 (10)	4.24 (1.90)
Exp2	654 (15)	164 (11)	3.99 (1.36)
Exp3	630 (24)	155 (15)	4.06 (1.60)
Exp4	598 (49)	143 (3)	4.18 (16.3)
Exp5	656 (21)	149 (12)	4.41 (1.75)
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	1299 (207)	185 (97)	7.02 (2.13)
Exp2	1337 (301)	207 (87)	6.46 (3.46)
Exp3	1316 (198)	204 (96)	6.45 (2.06)
Exp4	1443 (62)	201 (57)	7.18 (1.09)
Exp5	1361 (147)	192 (201)	7.09 (0.73)



(a) The shortest path and the expanded vertices in the nearest neighbor graph.

(b) The shortest path and the expanded vertices in the beamlet graph.

Figure 10: Example II: (a) The shortest path in the NNG for a 64×64 image; the yellow crosses denote the expanded vertices during the search; (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.

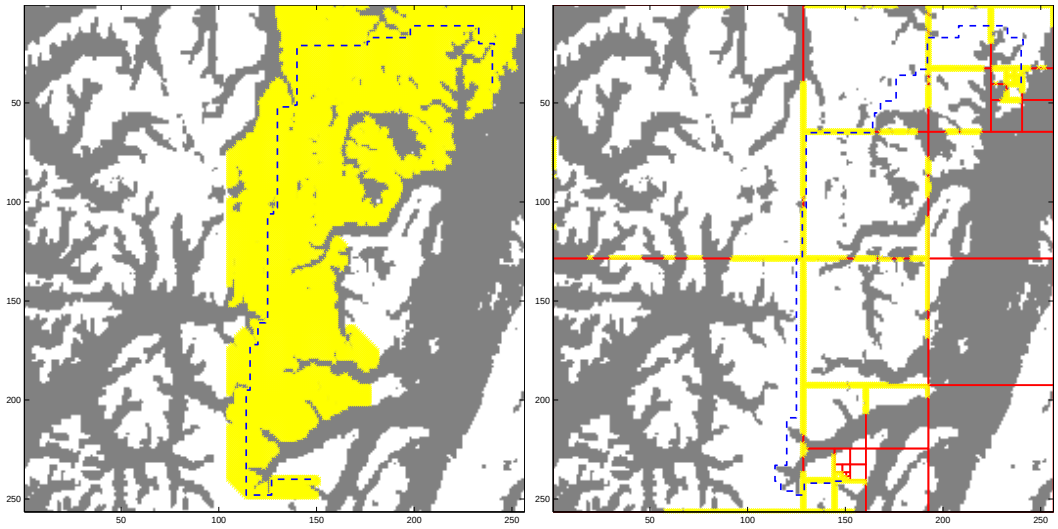
Table 3: Multiscale A* Algorithm Compared to Traditional A*. Example II.

ImageSize	64×64		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	1931 (307)	345 (260)	5.60 (1.18)
Exp2	2025 (130)	366 (87)	5.53 (1.49)
Exp3	2396 (234)	383 (159)	6.26 (1.47)
Exp4	2223 (267)	351 (46)	6.33 (5.80)
Exp5	2017 (181)	346 (143)	5.83 (1.27)
ImageSize	128×128		
GraphType	**NNG**	**BeamletGraph**	**Ratio**
Exp1	7254 (164)	666 (140)	10.89 (1.71)
Exp2	7755 (320)	814 (143)	9.53 (2.24)
Exp3	6712 (217)	742 (198)	9.05 (1.10)
Exp4	7755 (320)	814 (143)	9.52 (2.24)
Exp5	7588 (299)	770 (281)	9.85 (1.06)

The proposed beamlet-based graph structure was also tested on a real-world environment. Specifically, Figure 11 shows the elevation map of a certain area in the US. In the figure, gray areas are obstacles. The results of the shortest paths and the node expansions obtained using the nearest neighbor graph and the beamlet graph are shown in the same figure. For this scenario, the number of node expansions in the nearest neighbor graph and the beamlet graph are 16083 and 1043, respectively, which shows that by using the proposed graph structure, the speed of the shortest path-finding benchmark algorithm can be improved by an order of magnitude.

2.5.2 Comparison Using Stronger Heuristics

The beamlet graph can be viewed as a data structure that stores a more accurate heuristic than the basic L_1 heuristic. Thus, the superiority of m-A* over A* is the greatest when the baseline heuristic is the weakest. The situation may be quite different if the baseline heuristic is very accurate. So the question of whether m-A* can still outperform A* with a more accurate heuristic is relevant. Consequently, we also tested m-A* with a very accurate heuristic, namely, a true distance heuristic (TDH) with differential distance heuristic. The results from the comparison between



(a) The shortest path and the expanded vertices in the nearest neighbor graph.

(b) The shortest path and the expanded vertices in the beamlet graph.

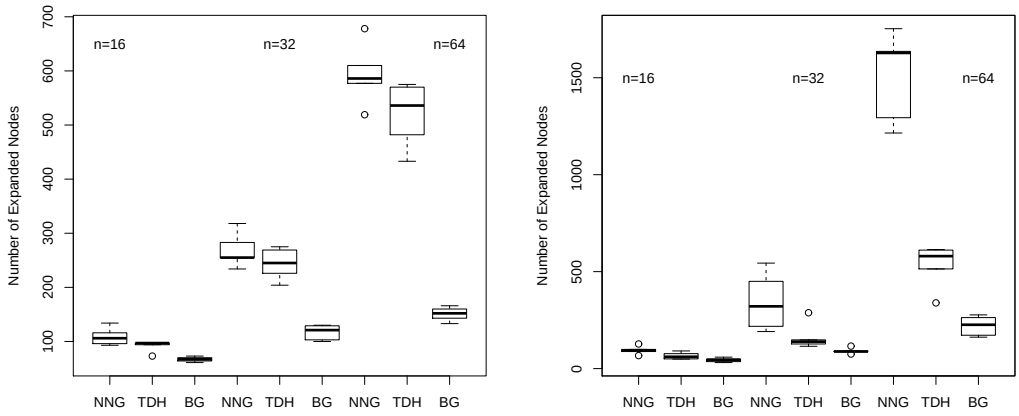
Figure 11: Example III: (a) The shortest path in the NNG for a 256×256 image; the yellow crosses denote the expanded vertices during the search; the blue dashed line shows the shortest path. (b) The shortest-path and the expanded vertices in the beamlet graph for the same image.

the standard A* (with L_1 distance heuristic), the A* with a true distance heuristic (A*-TDH) and the m-A* are shown in Figure 12. For each image size of $16\times16, 32\times32$ and 64×64, we randomly constructed five different gridworlds and run A*, A*-TDH, m-A* on each one respectively. As shown in these figures, m-A* gives the smallest number of node expansion among the three algorithms even when the stronger TDH heuristic is used. Furthermore, it is shown that the gain from m-A* is more significant as the image size increases.

2.6 Discussion and Related Prior Work

2.6.1 Beamlets as a Predecessor

The algorithm proposed in this chapter is rooted in the beamlet theory–a multiscale methodology for linear and curvilinear features in 2-D. Beamlets provide a framework for multiscale analysis, in which line segments play a role analogous to the role played by points in wavelet analysis. They add two crucial elements missing from



(a) Boxplot of comparison between A*, A*-TDH, m-A* for Example I.

(b) Boxplot of comparison between A*, A*-TDH, m-A* for Example II.

Figure 12: (a) Summary of numerical comparison between A* with L_1 heuristic, A* with TDH, and m-A* for Example I. Here n denotes the size of the gridworld. Each comparison group is derived based on numerical results from five randomly generated gridworlds.; (b) Summary of numerical comparison between A* with L_1 heuristic, A* with TDH, and m-A* for Example II. The notation is the same as in Example I.

wavelet processing, however: orientation and elongation information [18]. Beamlets are proven to achieve optimal asymptotic performance in feature detection problems [2]. They have been used to design efficient coding algorithms for images made of curves [42, 19]. Beamlets are numerically more efficient than traditional curve processing algorithms, such as JBIG2 [40], because they make use of their inherent multiscale structure in an innovative and efficient manner. The PFR-RDP used in this article is similar to the beamlet-decorated recursive dyadic partitioning of [18], depicted in Figure 13.

2.6.2 Related Work

Recent work on the shortest path finding problem bears some similarities with our methodology and thus a comparison is warranted. From the vast amount of the existing literature, we distinguish four algorithms that use multi-scale ideas to speed

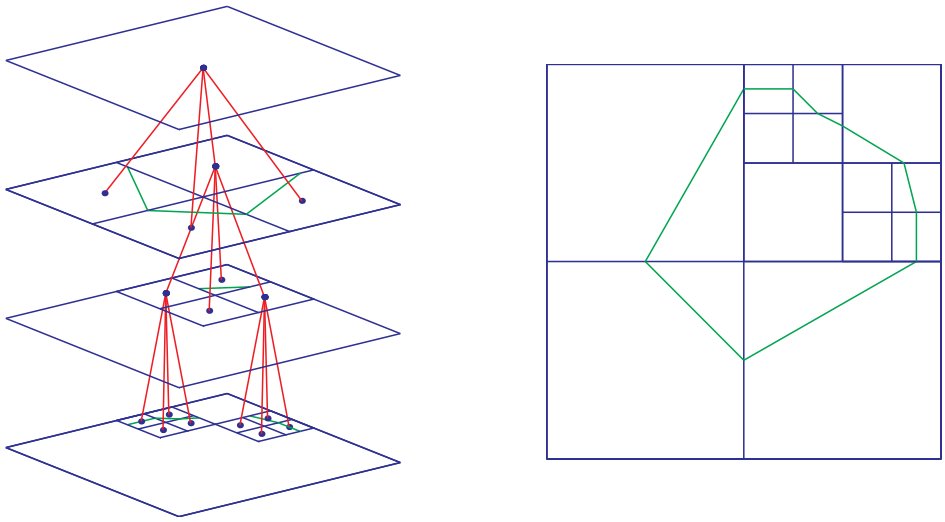


Figure 13: To each path in the beamlet graph in [18] corresponds a polygonal curve in the plane.

up the search during execution: state abstraction strategies (AS) [8, 69, 67], hierarchical A* (h-A*) [49], contraction hierarchies (CH) [30], and true distance heuristics (TDH) [22, 68].

Recall that the beamlet graph is induced from the nearest neighbor graph, and it has a reduced number of vertices. It takes advantage of "long-distance" interactions between vertices, which correspond to cells that may be very far apart in the original map. This is similar–in spirit–to the state abstraction strategy that has been intensively analyzed in [8, 69] and originally proposed in [67]. The strategy consists of building heuristic functions in an abstraction space. Actions in the abstraction space are then refined to actions in the environment by the A* algorithm, thereby substantially accelerating searching. Despite the apparent similarity between m-A* and the algorithms in [8] and [69] in the use of abstraction ideas, there are nonetheless distinct differences between the two approaches. First, their objectives are quite different. The objective of [8] is real-time execution and adaptive graph exploration; the goal is to determine the few next moves locally and quickly. Our objective, instead, is to find the shortest path globally. As a result, m-A* always guarantees an

optimal solution, while the AS in [8] only ensures sub-optimality. On the other hand, AS works with general graphs, and not only with gridworld graphs, which are the focus of the present chapter. Most closely related to m-A* is probably the sector abstractions (SA) of [69]. Sector abstractions are inspired from the hierarchical path finding A* (HPA*) algorithm of [6] and are also limited to grid-based maps. The key idea is to divide the map into clusters (corresponding to equally-sized squares) and generate a new graph by using the information of the free cells belonging to the boundary of these clusters. This bears a strong resemblance with m-A*. However, the beamlet-like connectivity is absent in [69, 6] and the vertices of the abstract graph are just a subset of the original gridded map. Furthermore, m-A* adopts a "bottom-up" fusion algorithm to organize pre-computed information based on the self-similarity across multiple resolution scales. This is missing from HPA* and SA.

The hierarchical A* (h-A*) algorithm of [49] builds a hierarchical abstraction of the state space based on the "clustering" of the "max-degree" vertex in the nearest neighbor graph. The process is repeated until all states (i.e., vertices) have been assigned to some abstract state. The heuristic used in the original graph is obtained via a shortest path search in the abstracted state space graph. Redundant expansion of nodes is avoided, since once the shortest path search from one vertex to the goal in the higher abstraction graph is completed, all the heuristics of the subsequent vertices in the path to the goal vertex, and its children on the next abstraction level are obtained without much difficulty (the latter is called "h*-caching" in [49]). The proposed beamlet-based graph also induces a hierarchical structure on the original graph, which is obtained, however, via path-finding on the PFR-RDP. The bottom-up fusion algorithm combines local information at the finer (lower) levels into global information at the upper (coarser) levels, while avoiding redundant calculations at each level, since only free boundary cells are involved at each step. Another difference between m-A* and h-A* is that for the gridworld graphs we consider, there is no

difference between the degrees of the vertices in the graph (that is, all vertices have the same degree owing to the use of a topological graph), which means that the PFR-RDP is a more appropriate tool for building the hierarchical information structure.

In [30] the concept of *contraction hierarchies (CH)* is proposed for path planning for large-scale road networks. In [70] it is shown that hierarchical abstractions and CH have similar overhead and performance. Contraction hierarchies is a particularly powerful technique for road networks. There are two basic steps involved in CH: (a) an ordering of all nodes, (b) the contraction, i.e., the removal of certain nodes based on the ordering, and the insertion of shortcut edges, so that the shortest paths are preserved when the contracted nodes are removed from the graph. CH works with any node ordering; however, the ordering has a huge influence on preprocessing and query performance. Although several alternatives exist, a bidirectional search is typically used on the contracted graph to carry out the search [31]. Although both CH and m-A* introduce hierarchies, the former is specifically designed for roadmaps, whereas the beamlet graph used in m-A* is more suitable for gridworlds. The induced PFR-RDP in m-A* takes advantage of the topological information in the gridworld, which does not exist in the CH framework. The ordering of vertices in CH is critical and quite complex, involving many considerations. The PFR-RDP used in m-A*, on the other hand, is intuitive and easy to implement. However, the PFR-RDP only makes sense when a gridworld formulation is adopted. Despite the similarities between CH and the beamlet graph (BG), neither of them is a special case of the other. For an $n \times n$ image the CH has n^2 levels, while the BG has $\log n$ scales. Therefore BG is a lot simpler. Furthermore, the bottom-up fusion algorithm of m-A* is also very different from the contraction steps used in CH. By design, it is expected that the bottom-up fusion algorithm should be much faster. In fact, as shown in Section 2.4.1, the complexity of the bottom-up fusion algorithm is $O(n^3)$. No order of complexity is known for the CH, due to the many possibilities in its realization. Note however

that CH would run contraction steps n^2 times. It is thus expected that, in the worst case, the contraction steps in CH will be more than $O(n^3)$ (given that there are n^2 nodes).

Other recent path-planning algorithms focus on the use of "optimal" distance heuristics to guide the search during the A* and thus speed up the graph search algorithm. Although the all-pair distance would be the perfect heuristic for searching the shortest path, the calculation and memory required would be unrealistic. The *true distance heuristic (TDH)* of [22, 68] reduces the memory requirements by using only a subset of the all-pairs-shortest-path information. To achieve this, the method predetermines the distance between k "landmark" nodes (where $k \ll n^2$)–also called canonical states in [22, 68]–on which the all-pair shortest paths algorithm runs. This enables a more precise estimate of the distance to the destination. The complexity of TDH can be easily verified to be $O(kN) = O(n^3)$, where $N = n^2$ is the number of vertices in the graph with corresponding memory requirements $O(kn^2) = O(n^3)$, if $k = O(\sqrt{n^2})$ (cf. Table 1 in [22]). That is, both TDH and the proposed algorithm share the same order of magnitude of computation and storage complexity. However, the success of the TDH depends crucially on the appropriate selection of the k canonical states. In fact, as mentioned in [22] "the best number and location of canonical states is an open problem." The boundary free cells obtained systematically via the PFR-RDP in m-A* can be viewed as canonical states. In that sense, the proposed PFR-RDP provides (at least) a partial answer to the previous question. Most importantly, our beamlet graph is built on multiscale information, which is not evident in the TDH approach.

It is important to mention that the proposed m-A* is not tied to any particular heuristic; any suitable heuristic from the literature will do. Instead, m-A* takes advantage of a simpler graph to perform the search—the beamlet graph. Therefore, although for the sake of simplicity, in the current implementation a unidirectional

search along with the L_1-distance heuristic was used, a bidirectional search could have been used instead without much difficulty, with all the conclusions remaining essentially the same. Note that many of the most recent work in advance path-planning [30, 31, 32] all employ bidirectional search during their execution of the A* algorithm.

Since the original submission of the chapter, new related work has been published including [33, 36], both of which incorporate similar aspects with the proposed graph search algorithm. The method in [36] identifies rectangular empty areas and prunes all interior nodes, leaving only the ones at the perimeter, similarly to the boundary cells of PFR-RDP in m-A*. The authors in [33] partition the original graph into disjoint subgraphs with few border nodes (the so-called "portals"). True distances between all pairs of portals are stored and used as admissible heuristics. This idea is similar to the "beamlet" connectivity between the boundary nodes of PFR-RDP used in m-A*.

2.7 Conclusions

We have introduced an innovative beamlet-based graph structure that facilitates path-finding in gridworlds with obstacles. The main idea is the construction of a graph structure that encodes efficiently long-distance interactions between vertices that go beyond the four-neighbor connectivity relations of the underlying nearest-neighbor topological graph. This is achieved by iterative subdivisions of the entire environment that give rise to a hierarchy of graphs, along with an innovative bottom-up fusion algorithm that combines local information from the lower (finer) scales to obtain global information at the upper (coarser) scales. Both the theoretical complexity analysis as well as the numerical examples demonstrate that the proposed multiscale A* (m-A*) algorithm provides significant improvements over the original A* algorithm applied on the original nearest neighbor graph.

The proposed graph structure can be viewed as the foundation on which several extensions of existing graph search algorithms can be based on. For instance, the multiscale graph structure in an incremental search setting would lead to multiscale versions of LPA* and/or D* algorithms [47, 48, 46, 65], allowing to handle dynamic changes in the environment. For some initial results towards this direction, see [53]. In case of motion-planning, beamlets can be used to efficiently and naturally incorporate curvature restrictions on the resulting paths. This is crucial for mobile agents with limited turning capability (e.g., fixed-wing UAVs, unmanned ground vehicles, etc). Also, the generalization of the proposed 2-D multiscale strategy to 3-D (or higher dimensions) is straightforward, albeit computationally more involved. Finally, the recursive nature of the bottom-up fusion algorithm invites the possibility for its parallelization. The potential of the parallel execution of the bottom-up fusion algorithm would tremendously improve the overall performance of m-A*.

CHAPTER III

AN INCREMENTAL, MULTI-SCALE SEARCH ALGORITHM FOR DYNAMIC PATH PLANNING WITH LOW WORST-CASE COMPLEXITY

3.1 Introduction

Dynamic path-planning deals with the solution of shortest-path problems on a graph, when the edge weights in the graph change over time. The Lifelong Planning A* algorithm (or LPA* for short) [47] is a well-known, widely used algorithm to solve dynamic path-planning problems, especially in mobile robotic applications. In numerical experiments it was observed however that LPA* can have unfavorable worst-case complexity. In particular, if a vertex located close to the original optimal path changes, a large number of vertex expansions may be required to replan the optimal path to the destination. In other words, LPA* is sensitive (i.e., not "robust"), in the sense that the number of required vertex updates can vary widely, depending on the location of the updated vertex in the graph. To demonstrate this point, consider the dynamic shortest-path problem illustrated in Figure 18. The objective in this figure is to find the shortest path while some of the vertices may become blocked over time. In Figure 19 (blue line), we observe that the number of vertex expansions in the LPA* varies significantly for this problem. Although most of the time, LPA* requires only a few expansions, in several noticeable occasions, the number of expansions is huge. Such a variation in the number of vertex expansions is not desirable, as it reveals unfavorable worst-case performance. Ideally, one would like the number of expansions to be relatively immune to the location of the updated vertices. Our objective is to introduce a modification of the LPA* that keeps the number of expanded vertices

approximately constant (compared to the classical LPA* implementation), regardless of the location of the updated vertex.

The main idea of the proposed *multiscale LPA* (m-LPA*)* algorithm is to utilize pre-computed multiscale information of the environment to formulate an associated search graph of smaller size, therefore reducing the computational complexity. The intuition behind the proposed algorithm is summarized as follows. Consider a uniform n-by-n grid representing the world (or an image) assuming, without loss of generality, 4-nearest-neighbor connectivity. Given a source and a destination, the search graph is abstracted from the environment before applying any path-planning search algorithm. When a change in the environment leads to the update of a certain vertex in the graph, the amount of computations required by LPA* during replanning varies dramatically, depending on the location of the updated vertex. Specifically, when the update happens to induce a "local dead-end" or if it is near the source, the number of vertex expansions can be extremely high. Figure 19 demonstrates this point. The proposed m-LPA* algorithm, on the other hand, takes advantage of multiscale information extracted from the environment and therefore reduces the computational complexity in both the initial planning and replanning steps simultaneously. This is achieved by making extensive use of a beamlet-like graph structure, which is based on a suitably pruned quadtree representation of the environment (called in the sequel the path finding reduced recursive dyadic partitioning, or PFR-RDP for short). The PFR-RDP encodes the information of the environment in a hierarchical, multiscale fashion, keeping track of "long-range" interactions between the vertices in the underlying beamlet graph. Therefore, when a vertex update that would induce a "local-dead-end" in the nearest neighbor graph takes place, this information is "transmitted" in the beamlet graph by modifying the hierarchical structure of the PFR-RDP to include new vertices and edges, so that no more dead-end exists in the new (beamlet) graph.

In the initial planning step, m-LPA* works exactly the same way as the previously proposed m-A* algorithm [54], by formulating a smaller-size search graph from the multiscale information obtained from a specifically designed bottom-up fusion algorithm (see Algorithm 5 in Section 3.4.2). This leads to a significant reduction in computational complexity (see Section 3.3 for a brief review of the A* and m-A* algorithms). When a vertex is updated, m-LPA* will further decompose the area where the updated vertex is located and will update the graph before replanning. The number of vertex expansions during the replanning step is therefore reduced, owing to the smaller-size search graph obtained in the initial planning.

The theoretical analysis of the associated computational complexity reveals that, in the worst case, the proposed algorithm has a lower order of complexity than the LPA* algorithm. In our numerical experiments, it was found that the m-LPA* algorithm can dramatically reduce the number of vertex expansions, in the worst case. To ensure a fair comparison, the implementation of both LPA* and m-LPA* algorithms was based on Fibonacci heaps, which is known to be one of the most efficient data structures for sorting problems [11]. We believe that a comparison using Fibonacci heaps is more accurate than using, say, binomial heaps [47]. In addition, the use of Fibonacci heaps allows one to find the minimum vertex in the priority queue faster. Based on these numerical studies, it is shown that the proposed method is very stable when tested under several different scenarios.

The proposed algorithm belongs to the general class of multiscale/multiresolution, dynamic path-planning algorithms. Multiresolution decomposition techniques for path-planning have been used extensively in the literature. See, for example, [45, 3] and, more recently, Refs. [57] and [72, 12] where wavelets are used to create several levels of abstraction for the environment. The approach in [72, 12] uses a higher resolution close to the agent where is needed most, and a coarser resolution at large distances from the current location of the agent. The motivation for this approach

stems from the fact that the agent's immediate reaction to an obstacle or a threat is needed only in the vicinity of its current position. Faraway obstacles or threats do not have a great impact on the vehicle's *immediate* motion. Therefore, it makes sense from a computational point of view to generate a solution with greater accuracy only locally, around the current location of the agent, with decreasing resolution further away. Multiresolution strategies are indispensable for on-line implementation when the agent has limited computational resources. In this chapter we go beyond the use of wavelets to encode the environment information across different scales. Although wavelets are very efficient in that respect, they also have some drawbacks. Most importantly, they lack orientation information and can be roughly characterized as isotropic. Beamlets provide a framework for multiscale analysis, in which line segments play a role analogous to the role played by points in wavelet analysis. They add two crucial elements missing from wavelet processing, however: orientation and elongation information. They provide an optimal way to approximate curvelinear features in 2D. Beamlets connect points that may be far apart, thus encoding "far-away" interactions in the environment. In this chapter we show that multiresolution/multiscale strategies based on beamlet-like ideas, can also lead to increased computational robustness at the execution level.

The rest of the chapter is organized as follows. Section 3.2 formulates the path-planning problem in a dynamically changing environment. Section 3.3 reviews the multiscale A* algorithm and the LPA* algorithm. These two algorithms are the foundation of the m-LPA* algorithm proposed in this chapter. Section 3.4 offers the details of the proposed m-LPA* algorithm. Its complexity analysis is conducted in Section 3.5. Section 3.6 provides numerical examples to compare m-LPA* with LPA*, while Section 3.7 compares m-LPA* with other closely related search algorithms, pinpointing their apparent similarities and their main differences. Finally, Section 4.7 summarizes the findings of the chapter and suggests some possible future extensions.

3.2 Problem Formulation

In a path-planning problem we are given a graph $G = (V, E)$, with vertex set V and edge set $E \subseteq V \times V$. For instance, V is typically the set of possible vehicle locations and E represents transitions between these locations. The weight of each edge denotes the cost of transitioning between the two locations represented by the vertices connected by the corresponding edge. Planning a path in G can thus be cast as a single-pair shortest path problem on this graph.

In a deterministic environment, a graph has constant edge weights over time. Dijkstra's algorithm and the A* algorithm and their numerous variants are classical methods to compute optimal paths to the destination from every location or from a single location in the graph, respectively. In many practical applications (especially in the area of robotic mobile vehicles) it is common to obtain updated information about the environment over time. This leads to a replanning problem on a graph with changing edge weights. If the replanning is done from scratch (that is, without using any prior information of the graph structure or the edge weights), this may result in wasted time and/or computational resources. Therefore, several efficient algorithms have been developed in the literature to adjust the current shortest path when a change takes place, without performing redundant calculations. References [47] and [48], in particular, proposed the Lifelong Planning A* (LPA*) algorithm which considers the dynamic path-planning problem with fixed source and destination. Another popular replanning algorithm for a vehicle navigating in a dynamically changing environment that combines heuristic and incremental searches is D* (or dynamic A*) [65]. In [46] the authors extended the LPA* algorithm and proposed the D*-Lite algorithm, which is similar in spirit, but much simpler, than D*. In many applications, the curvature of the resulting path is also of interest in order to guarantee kinematically feasible vehicle trajectories. In this context, references [23, 24, 56, 13] incorporate curvature information to make the planned route as realistic as possible.

In this chapter, we consider a path-planning problem in a dynamically changing 2-D environment. Without loss of generality, it is assumed that the environment can be represented by an n-by-n square image, where n is dyadic, i.e., $n = 2^J$ and J is a positive integer. Note that such an image-based formulation is quite common in the path-planning literature [23]. As usual, it is assumed that the image contains two types of pixels: gray pixels (representing non-traversable obstacles) and white pixels (representing traversable free cells). The path-planning problem is to find the shortest path between a given pair of *source* and *destination* pixels. Under this binary image assumption, a change in the environment is formulated as a change in the *traversability properties* between certain cells[1]. The task of replanning then consists of finding another shortest path inside the new environment, while avoiding a large number of further vertex expansions.

Among the several existing replanning algorithms mentioned previously, in this chapter we focus exclusively on LPA* since it represents a widely used, state-of-the-art algorithm in the area of incremental replanning. It is reminded that LPA* operates essentially the same as the well-known A* algorithm in the initial planning step but, in addition, it utilizes the concept of "local inconsistency" of vertices to control the size of the priority queue and hence the number of vertex expansions.

As mentioned earlier, numerical examples indicate that the number of vertex expansions in the LPA* algorithm can vary significantly depending on the obstacle location. Specifically, in several cases when the blocking vertex happens to be somewhere along the initial shortest-path resulting in a "local dead end," or when there are too many obstacles in the environment, the number of vertex expansions in the LPA* replanning part can blow up. We want to avoid such undesirable behavior. Our objective is to devise an algorithm that retains the nice properties of LPA*, while

[1] A cell is defined to be a collection of pixels in the environment. At the highest resolution, cell and pixels are equivalent.

at the same time it maintains an almost constant number of vertex expansions when the environment is changed. The proposed multiscale LPA* (m-LPA*) algorithm takes advantage of the sparse information induced by the quadtree decomposition in a hierarchy of dyadic squares. This is the same technique adopted in our previously proposed *multiscale A* (m-A*)* algorithm [54]. As shown in Section 3.5, such a strategy can reduce the number of vertex expansions significantly, and therefore also reduce the overall worst-case replanning complexity.

3.3 The Multiscale A and Lifelong Planning A* Algorithms*

In this section we briefly summarize the key points of the multiscale A* algorithm (m-A*), which lay the foundation for m-LPA*. The m-A* algorithm is an extension of the classical A* algorithm; it takes advantage of preprocessed multiscale information in order to reduce the overall computational complexity. Afterwards, we provide a brief overview of the LPA* algorithm. The LPA* is an incremental search algorithm that replans the initial path when a vertex update occurs owing to a change in the environment. These two algorithms (the m-A* and the LPA*) are the precursors of m-LPA*.

3.3.1 The Multiscale A* (m-A*) Algorithm

The classical A* algorithm searches through all free cells in the environment, which can be overwhelmingly redundant. The motivation for the multiscale-A* (m-A*) algorithm is to construct a smaller size search graph, on which the computational complexity of searching for the shortest path is significantly reduced. The m-A* algorithm is based on the following key elements:

(i) The *Recursive Dyadic Partition* (RDP) and its extension, the *Path-Finding Reduced RDP* (PFR-RDP). The elements of (PFR-)RDP are the dyadic *d-squares*, parameterized by *scale* and *location*. For the single-pair, shortest path-planning

problem, the PFR-RDP is first constructed, and all free cells at the boundaries of each d-square in the PFR-RDP are selected as the vertices in the search graph. Figure 14 shows the d-squares for the shown partition, along with the additional boundary cells used as vertices in the search graph.

(ii) The idea of *beamlet-like connectivity*. This provides an extension of connectivity between a pair of non-adjacent free cells. Within each d-square, besides the assumption of 4-nearest-neighbor connectivity, we further consider any pair of free boundary cells to be connected if there exists an obstacle-free path between the two, which lies within that d-square. This shortest path is called a *beamlet*. Readers may notice that this is a generalization of the beamlet concept introduced in [18]. Therein, the beamlets are defined as straight line segments of variable length, scale and angle, connecting boundary cells of d-squares. They have been applied successfully to image processing applications (i.e., edge detection). Please see [16, 17], and [19] for more details. The *beamlet graph* is defined to be the search graph with these two types of connectivity.

(iii) The *bottom-up fusion algorithm* designed to obtain the edge weights of the search graph from different scale dyadic squares. The algorithm is a recursive method that employs the RDP in each d-square from the PFR-RDP, in order to compute the inter-distances between the free boundary cells for all d-squares in the environment. In other words, the bottom-up fusion algorithm finds all shortest paths between the free boundary cells, i.e., the edge weights in the beamlet graph. The main idea of the algorithm is based on the observation that if we know the inter-distances between the free boundary cells within each of the smaller d-squares, and by considering the connectivity of the free boundary cells that belong to neighboring d-squares, we can treat all free boundary cells of the

four d-squares at the next scale as vertices in a "fused" graph. The distance between free cells from neighboring d-squares can be defined by direct neighbors. Figure 15 shows how this "fusion" of distance can be conducted recursively. The pseudo-code of the bottom-up fusion algorithm is given in Algorithm 5.

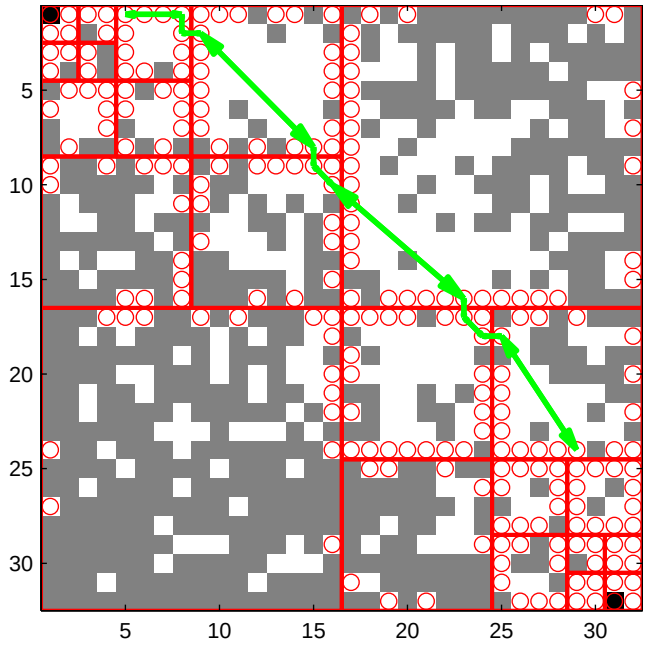


Figure 14: Illustration of the Path-Finding Reduced Recursive Dyadic Partition (PFR-RDP) on a 32×32 image. The black cells are the source and destination. The red circles denote the free boundary cells, i.e., the vertices in the beamlet graph. The green arrows show the edge weights between two free cells constructed via the bottom-up fusion algorithm across several scales in the PFR-RDP.

The combination of beamlet-like connectivity and multiscale decomposition in m-A* can reduce the depth of the search tree from $O(n)$ roughly to $O(\log n)$, without increasing the branching factor in each layer. The beamlet graph thus has $O(n)$ vertices and $O(n^2)$ edges. The worst-case complexity of running A* on the beamlet graph is therefore $O(n^2)$, compared to $O(n^2 \log n)$ when using the 4-nearest-neighbor graph.

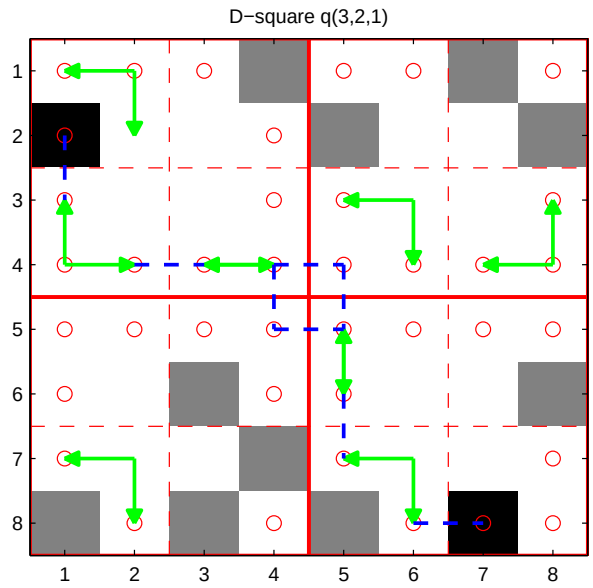


Figure 15: Magnified view of the fusion process in the d-square q(3,2,1) of Figure 14. The fusion is conducted in a complete dyadic partition. Notice that the solid red grid stands for the partition corresponding to the second layer of the associated quadtree and the dashed red grid corresponds to the partition with respect to the third layer. The green arrow lines indicate the inter-distance connectivity between the corresponding free boundary cells. The blue lines show the fusion process.

3.3.2 Incremental Search Algorithm: LPA*

Heuristic search methods are likely to find the shortest path faster than uninformed search methods. Incremental search methods, on the other hand, promise to find shortest paths by solving a series of similar path-planning problems faster than it is possible by solving each path-planning problem from scratch. The *Lifelong Planning A** (LPA*) in [47] can be viewed as an incremental version of the A* algorithm, the latter being a heuristic enhancement of the well-known Dijkstra algorithm. The LPA* repeatedly finds shortest paths from a given source to a given destination, while the edge weights of the graph change, or while vertices are added or deleted. The first search of LPA* is the same as that of the classical A* algorithm. Subsequently, the algorithm breaks ties in favor of vertices with a smaller g-value (i.e., the current estimated distance from the start vertex). As a result, many of the subsequent searches are potentially faster, because the algorithm reuses those parts of the previous search

graph that are identical to the new one.

To facilitate the subsequent discussion on the proposed multiscale version of LPA*, we first need to introduce some notation as follows. Let $Succ(v) \subset V$ denote the set of successors of the vertex $v \in V$, let $Pred(v) \subset V$ denote the set of predecessors of vertex v, and let $c(v, v')$ denote the cost of moving from vertex v to vertex v'. Let also $h(v, v_{\text{goal}})$ denote the heuristic that guides the search direction to the goal destination. Finally, let $g(v)$ denote the start distance of vertex v, that is, the length of the shortest path from v_{start} to v.

The LPA* algorithm uses two estimates of the start distance, namely, $g(v)$ and $rhs(v)$. The rhs-values are the one-step lookahead values based on the g-values for each vertex, and thus they are potentially better informed than the g-value. Specifically, $rhs(v) = \min_{v' \in Pred(v)}(g(v') + c(v', v))$. A vertex is defined to be *locally consistent* if and only if $g(v) = rhs(v)$; otherwise it is said to be locally inconsistent. It should be clear from the above definitions that all vertices are locally consistent if and only if their g-values are equal to their respective start distances. LPA* also maintains a priority queue, which always contains exactly the locally inconsistent vertices. The priority of vertices in the queue is based on the key value, which is defined to be $k(v) = [k_1(v), k_2(v)]$, where $k_1(v) = \min(g(v), rhs(v)) + h(v, v_{\text{goal}})$, and $k_2(v) = \min(g(v), rhs(v))$. The keys of the vertices in the priority queue roughly correspond to the f-values used by A*. LPA* always recalculates the g-value of the vertex (i.e., expands the vertex) in the priority queue with the smallest key value. This is similar to A*, which always expands the vertex in the priority queue with the smallest f-value. LPA* keeps expanding the vertices until v_{goal} is locally consistent and the key of the vertex to be expanded next is no less than the key of v_{goal}. This is similar to A*, which expands vertices until it expands v_{goal}, at which point the g-value of v_{goal} equals its start distance, and the f-value of the vertex to be expanded next is no less than the f-value of v_{goal}. If $g(v_{\text{goal}}) = \infty$ after the search, then there

is no finite-cost path from v_{start} to v_{goal}.

Note that LPA* does not make every cell locally consistent. Instead, it uses an informed heuristic to focus the search and the subsequent updates only on the vertices whose g-values are relevant for finding the shortest path. This is the main principle behind LPA*, and this is what makes LPA* a very efficient replanning algorithm.

*3.4 Multiscale Strategy in Dynamic Path Planning: m-LPA**

The proposed multiscale Lifelong Planning A* (m-LPA*) algorithm is an extension of the previously proposed m-A* algorithm [54] for the case of a dynamically changing environment. Recall that we consider a discrete 2-D environment of dimension $n \times n$ containing only *obstacles* and *free cells*. Each free cell is a vertex in the underlying topological graph and is connected to the free cells among its four nearest-neighbors. We assume, without loss of generality, that each edge has unit weight. We describe our approach in three steps. In Section 3.4.1, we describe the *Dynamic Path-Finding Reduced Recursive Dyadic Partition* (DPFR-RDP) step. This serves as the starting point of the proposed replanning scheme. We then describe how to update the beamlet graph for the purpose of replanning when a change that alters the traversability of the original path takes place. This is done in Section 3.4.2. The approach hinges upon the bottom-up fusion algorithm that collects the multiscale information of the graph. Section 3.4.3 presents the proposed multiscale LPA* algorithm, which essentially runs LPA* iteratively on the aforementioned updated beamlet graph.

3.4.1 Dynamic Path-Finding Reduced Recursive Dyadic Partition

We define two types of recursive dyadic partitions (RDP), namely, the complete RDP and the path-finding reduced RDP (PFR-RDP). The PFR-RDP is a partial recursive partition in the sense that not all d-squares are partitioned to the finest level. Figure 14 shows an example of a PFR-RDP on a 32-by-32 image.

Algorithm 4 DPFR-RDP (Dynamic Path-finding Reduced Recursive Dyadic Partition)

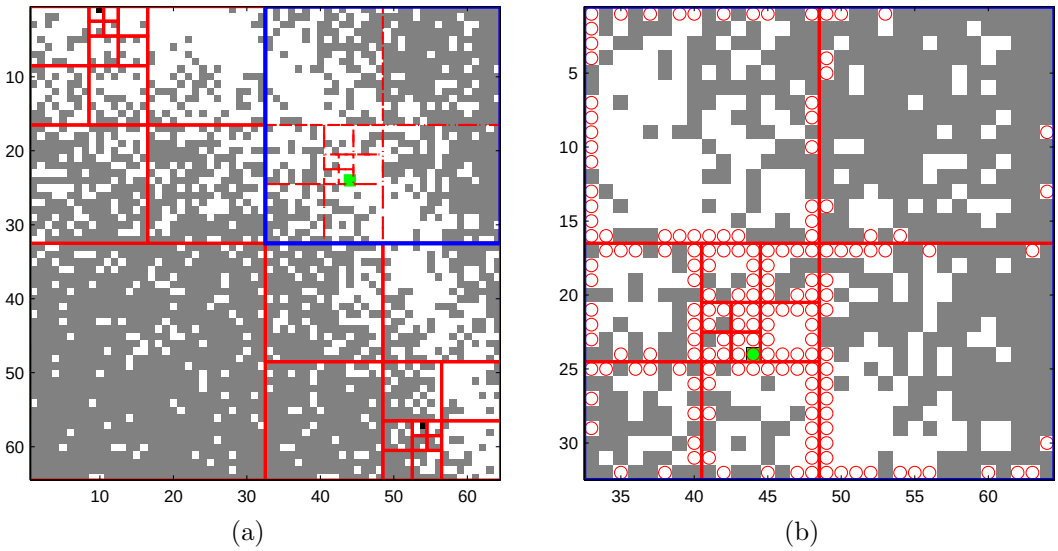
Set the largest scale to $J = \log_2 n$, where the image size is n by n.
Initialize the list *dptree* = [1, 1, 1]–the d-square at the coarsest level.
for $s = 1 : J - 1$ **do**
    For d-square at scale s in *dptree*
    if v_s (source) or v_e (destination) is in this d-square **then**
        In *dptree*, remove the line corresponding to this d-square;
        Partition into four equal, smaller d-squares, and insert them as new lines in *dptree*
    end if
end for
if v' (update) $\neq \varnothing$ **then**
    Locate the d-square in PFR-RDP where v' locates, denoted as $[s_{v'}, a_{v'}, b_{v'}]$
    Conduct PFR-RDP (which is the first **for** loop above) on $[s_{v'}, a_{v'}, b_{v'}]$ by setting $v_s = v_e = v'$
end if
return *dptree*

In order to make efficient use of the multiscale information in a dynamically changing environment, we extend this recursive dyadic partition to obtain a dynamic version of PFR-RDP: the Dynamic PFR-RDP (DPFR-RDP). This is just one more step in the construction of the PFR-RDP, in the sense that when a certain cell in the gridworld suffers from a traversability change, we first identify the d-square in the PFR-RDP in which this candidate cell is located, and we then conduct a further partial dyadic partition (only) in this candidate d-square. Figure 16 shows the DPFR-RDP for a 64-by-64 image. The pseudo code for the DPFR-RDP is given in Algorithm 4.

The DPFR-RDP enables us to obtain high resolution information around the cells in the environment that have changed. All free boundary cells for each d-square obtained from further dyadic partitioning are included as new vertices in the beamlet graph. This is the vertex update step of the beamlet graph.

3.4.2 Update of Multiscale Information in the Beamlet Graph

In order to update the edge weights in the beamlet graph that were influenced by the updated cell, it would be far more redundant if we were to recalculate from scratch



(a) (b)

Figure 16: (a) Illustration of the DPFR-RDP. The red grid shows the original PFR-RDP before any vertex update; the dashed red grid stands for the further partitioning after the green cell has been updated. The blue frame encloses the candidate d-square that is selected for further dyadic partitioning. (b) Magnified upper-right candidate d-square in (a). The free boundary cells are indicated by red circles. Except for one of the smallest newly-generated d-squares, all other d-squares contain the same inter-distance information as before, which is obtained through the bottom-up fusion process.

the inter-distances between all free boundary cells. In fact, this information has already been obtained during the bottom-up fusion process when we run m-A* at the initialization step (later on we show that LPA* runs exactly the same way as m-A* before the updates).

By taking advantage of the hierarchical inter-distance structure via the bottom-up fusion algorithm, the edge weights in the beamlet graph can be updated promptly. For instance, in Figure 16(b), except for the smallest d-square where the green cell is located, no change of edge weights happens in any other d-square. Only the updates of the edge weights in the smallest d-square that contains the updated cell need to be recalculated, which is trivial, given the fact that the finest scale d-square contains only four cells. The main effort thus involves running an all-shortest path algorithm on the graph constructed from all the free boundary cells of the newly added d-squares during the further partitioning. More generally, multiple updates at the same time can be processed the same way. Figure 17 shows a typical example of multiple updates performed simultaneously.

3.4.3 LPA* Algorithm on the Beamlet Graph

Now that we have established a method for updating the beamlet graph, we turn our attention to finding the shortest path on this graph. Given the source and destination, the first step of m-LPA* is to run m-A*. This gives us the beamlet graph and the initial optimal path. If there is no change in the environment, the algorithm terminates. When any cell is updated by some event, the dynamic PFR-RDP is constructed, and the multiscale inter-distance information obtained from the bottom-up fusion algorithm during the first step is used to update the vertices and edge weights in the beamlet graph. The final step is to run the LPA* algorithm on the updated beamlet graph. The whole algorithm is summarized in Algorithm 6.

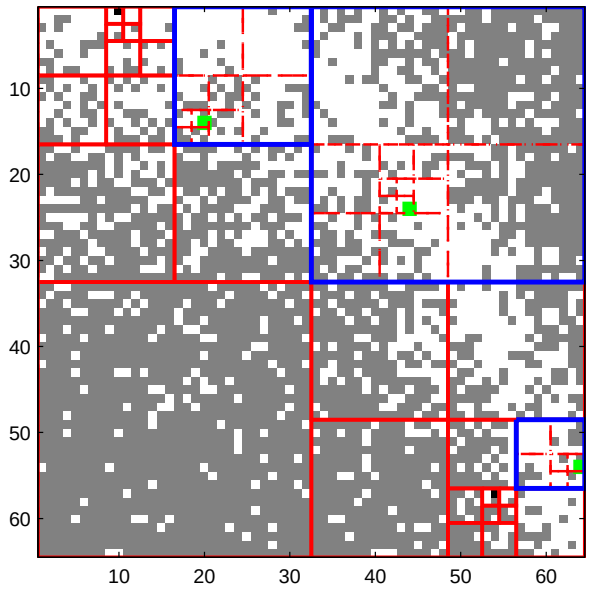


Figure 17: Illustration of processing multiple updates simultaneously. The red grid shows the original PFR-RDP obtained from m-A*; the dashed red grid stands for the further partitioning after the green cells have been updated. The blue frame encloses the candidate d-squares marked for further partitioning.

Algorithm 5 BottomUpFusion (For each d-square)

Read the parameters of each d-square: s (scale), a, b (location);
if $s = \log_2 n - 1$ **then**
    Compute the free boundary cells as vertices (Trivial case: only four cells in the d-square)
    Calculate the four nearest neighbor connectivity (edges) within each d-square
    Run Johnson's algorithm on the resulting graph to obtain all pairs of shortest paths: *cgraph* and *pathList*.
end if
if $s > 1$ **then**
    [*graph1, path1*] = BottomUpFusion$(s+1, 2a-1, 2b-1)$
    [*graph2, path2*] = BottomUpFusion$(s+1, 2a, 2b-1)$
    [*graph3, path3*] = BottomUpFusion$(s+1, 2a-1, 2b)$
    [*graph4, path4*] = BottomUpFusion$(s+1, 2a, 2b)$
    Merge *graph1, ..., graph4* into *Graph* by adding the connected edges between neighboring d-squares
    Run Johnson's algorithm on *Graph* and get *cgraph* and *tmpPathList*
    Insert the missing parts of paths in *tmpPathList* from *path1,..., path4* to obtain *pathList*
    return *cgraph, pathlist* (i.e., the beamlet graph)
end if

Algorithm 6 Multiscale Lifelong A* (m-LPA*)

1: **procedure** $Key(v)$
2: **return** $[g(v) \wedge rhs(v) + h(v_s, v); g(v) \wedge rhs(v)]$

3: **procedure** Initialize()
4: $OPEN = \emptyset$;
5: for all $v \in V$ $rhs(v) = g(v) = \infty$;
6: $rhs(v_s)$=0;
7: insert v_s with $Key(v_s)$ into $OPEN$

8: **procedure** UpdateState(v)
9: **if** $v \neq v_s$ **then**
10: $rhs(v) = \min_{v' \in Pred(v)}(c(v, v') + g(v'))$
11: **end if**
12: **if** $v \in OPEN$ **then**
13: remove v from $OPEN$
14: **end if**
15: **if** $g(v) \neq rhs(v)$ **then**
16: insert v into $OPEN$ with $Key(v)$
17: **end if**

18: **procedure** ComputeShortestPath()
19: **while** $\min_{v \in OPEN}(key(v)) < key(v_{\text{goal}}) || rhs(v_{\text{goal}}) \neq g(v_{\text{goal}})$ **do**
20: remove state v with min key from $OPEN$;
21: **if** $g(v) > rhs(v)$ **then**
22: $g(v) = rhs(v)$;
23: for all $v' \in Succ(v)$ UpdateState(v');
24: **else**
25: $g(v) = \infty$;
26: for all $v' \in Succ(v) \cup \{v\}$ UpdateState(v');
27: **end if**
28: **end while**

29: **procedure** Main()
30: Initialize *img*, v_s, v_e;
31: Conduct *PFR-RDP* and obtain *dptree*
32: Run the *Bottom-Up Fusion* algorithm on each d-square in *dptree* and get beamlet graph
33: **for** ever **do**
34: ComputeShortestPath();
35: Wait for changes in edge costs;
36: quadtree = FurtherPartition();
37: Update beamlet graph via *Bottom-up Fusion*;
38: **for** all directed edges (u,w) with changed cost **do**
39: Update edge cost $c(u, w)$;
40: UpdateState(u);
41: **end for**
42: **end for**

3.5 Complexity Analysis and Data Structure

In this section we discuss the cost of replanning and the influence of the data structure used to maintain the priority queue on the overall computational complexity of the m-LPA* algorithm. Roughly speaking, complexity of any search algorithm depends on two factors: the number of vertex expansions, and the number of heap percolations. In Section 3.5.1 the worst-case scenario analysis of the algorithmic complexity in terms of vertex expansions is provided, lending support to the benefits of the proposed m-LPA* algorithm. Section 3.5.2 considers the implementation issues to reduce the heap percolation overhead.

3.5.1 Worst-Case Complexity Analysis

In order to investigate the complexity of the replanning step, and without loss of generality, we assume that only one vertex update occurs every single time. Let us denote the number of vertex expansions as V_e. We have the following theorem.

Theorem 3.5.1 *In the worst case,* $|V_e| = O(n^2)$ *on the nearest neighbor graph, and* $|V_e| = O(n)$ *on the beamlet graph.*

Proof. In the worst case, the complexity of replanning has the same order as the initial path-planning [47], and hence $|V_e^{\mathrm{NNG}}| = O(n^2)$, where $|V_e^{\mathrm{NNG}}|$ denotes the number of vertex expansions for the nearest neighbor graph.

In the replanning part, multiscale information has already been obtained via the bottom-up fusion algorithm. There are two scale-1 d-squares and six scale-s d-squares when $s \geq 2$. Furthermore, for a d-square at scale s, there are at most $n2^{2-s}$ free boundary pixels. Therefore, the initial beamlet graph has $|V_1| \leq 2 \times 2n + 6 \times n + 6 \times \frac{n}{2} + 6 \times \frac{n}{4} + ... = 4n + 6n + 3n + \frac{3}{2}n + ... \leq 16n$ vertices. Furthermore, the number of vertices added during the replanning step has an upper bound of $|V_2| \leq 3 \times n + 3 \times \frac{n}{2} + 3 \times \frac{n}{4} + \cdots = 6n$. This is because during further partitioning of

the d-square where the update of a vertex takes place, we have three new d-squares at each scale (see Fig. 14 for an example). Hence, the total number of vertices during replanning is $|V| = |V_1| + |V_2|$, which is bounded by $|V| \leq 22n$. Thus, in the worst case, the number of vertex expansions in the replanning for the beamlet graph is of order $O(n)$.

3.5.2 Fibonacci vs Binomial Heap Implementation

An efficient data structure is required in order to maintain the priority queue and to find the minimum cost vertex at each step of the search algorithm with the least effort. A Fibonacci heap is used in m-LPA* to maintain the priority queue, instead of a binomial heap (used in [47]), because the Fibonacci heap has a better amortized running time than the binomial heap. During the search step of the LPA*, there are mainly three operations that change the content of the heap: insert, find-minimum, and delete-minimum. The complexity of these operations is important because they are used to calculate the heap percolation, which is a metric for comparing the performance of algorithms. It is computed as the sum of the total number of operations that maintain the heap structure (i.e., number of swapped parent and child pairs in the heap). In a Fibonacci heap, the operations *insert* and *find-minimum* work in constant (i.e., O(1)) amortized time, while the operation *delete-minimum* works in $O(\log n)$ amortized time [25]. In a binomial heap, on the other hand, the complexity of these operations is the same for all, namely $O(\log n)$. Hence, the heap percolation for the binomial heap is $(n_i + n_d + n_f)O(\log n)$, whereas using a Fibonacci heap the heap percolation is $(n_i + n_f)O(1) + n_d O(\log n)$. Here n_i, n_d and n_f denote number of insert, delete-minimum and find-minimum operations. This shows that the Fibonacci heap is a better choice for the data structure implementation in a replanning algorithm in terms of heap percolations.

Also note that the core component of m-LPA* is the beamlet graph obtained from

the preprocessed multiscale information, which has a reduced number of vertices, but an increased number of insertion operations, due to the generalization of connectivity between non-adjacent cells. As a result, the operation complexity of m-LPA* is dominated by insertion operations, which is approximately of order $O(1)$, compared to that $O(\log n)$ of LPA*. This observation also justifies the advantage of m-LPA* over LPA* in terms of heap percolations.

3.6 Simulation Studies

In this section we provide numerical experiments from several different scenarios, comparing the m-LPA* with the original LPA* algorithm. For each scenario, five randomly generated gridworlds are constructed, on which the comparison between m-LPA* and LPA* takes place. In the first scenario, we created an n-by-n image, assuming that the probability of a certain cell (indexed as (x, y), where $1 \leq x, y \leq n$) to be obstacle-free is given by $p(x, y) = \exp(-\gamma|\varphi(x, y)|)$, where the constant γ will be specified later. The intuition behind this model is that pixels (i.e., fine resolution cells) near the curve defined by $\varphi(x, y) = 0$ have higher probability to be free than the pixels far away from this curve.

In all numerical experiments, the total number of vertex expansions (V_e, the number of updates of the g-value of the vertices) is used as the metric to compare the efficiency of the two algorithms. We did not use heap percolation or CPU time, because CPU time is a machine-dependent metric, and heap percolation is approximately of order $O(1)$ per vertex expansion, as a direct result of using both a Fibonacci heap and a beamlet graph (see Section 3.5.2 above).

First, we compare the shortest paths and the number of expanded vertices during the initial planning step of LPA* and m-LPA*. Figure 18 shows the results for one of the sample environments for the case when $\varphi(x, y) = y - x^2/n$. As seen in these plots, the number of expanded vertices during the initial planning step using the beamlet

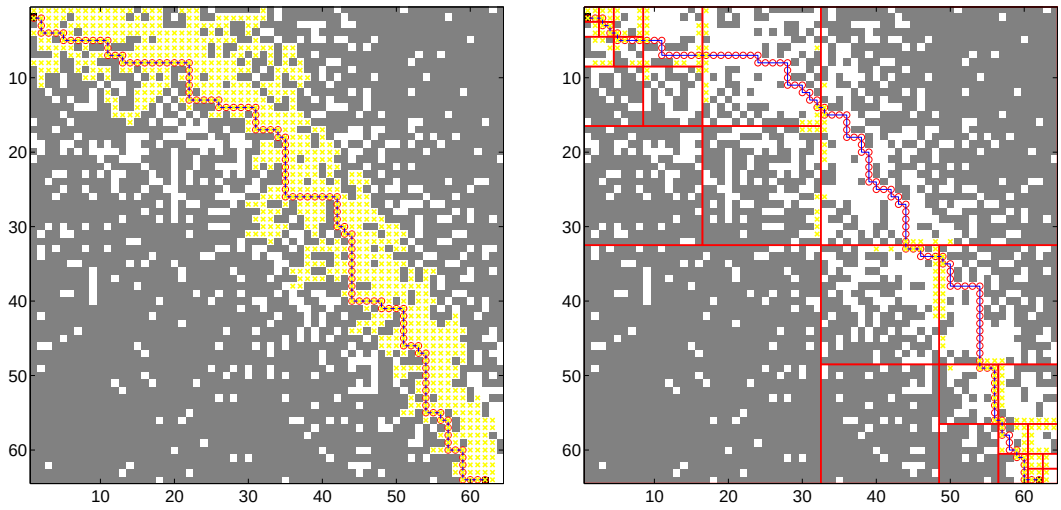


Figure 18: (a) The shortest path identified by the first step of LPA* in the nearest neighbor graph. The black cells are the source and destination, whereas the gray cells stand for the obstacles. The red sequence of circles denotes the shortest path. The yellow crosses denote the expanded vertices in the initial planning. (b) The shortest path identified by the first step of m-LPA* in the beamlet graph.

graph is much smaller than the one using the nearest neighbor graph. Multiscale information used in the initial step of planning significantly reduces the number of vertex expansions, and because of the use of Fibonacci heaps, the number of vertex expansions is the only step that is time-consuming.

Next, the number of vertex expansions during the replanning step of both LPA* and m-LPA* algorithms are compared. To this end, recall that the nearest neighbor graph and the beamlet graph are the underlined graphs in the LPA* and m-LPA* algorithms, respectively. Based on the intuition that when the updated vertex does not belong to the shortest path identified by either the LPA* or the m-LPA* algorithm, the number of vertex expansions tends to be small, in these examples we imposed the blocking vertex to belong in the set of vertices of the initial shortest path, except for the source and destination vertices.

For the scenario shown in Figure 18, we conducted five experiments with randomly generated gridworlds. For each experiment, the updated vertices were on the initial shortest path and they were updated one-at-a-time sequentially. Figure 19 shows the

pattern of the number of vertex expansions for LPA* and m-LPA* for one of these numerical experiments, but the pattern was consistent in all five experiments. The following observations are evident from Figure 19:

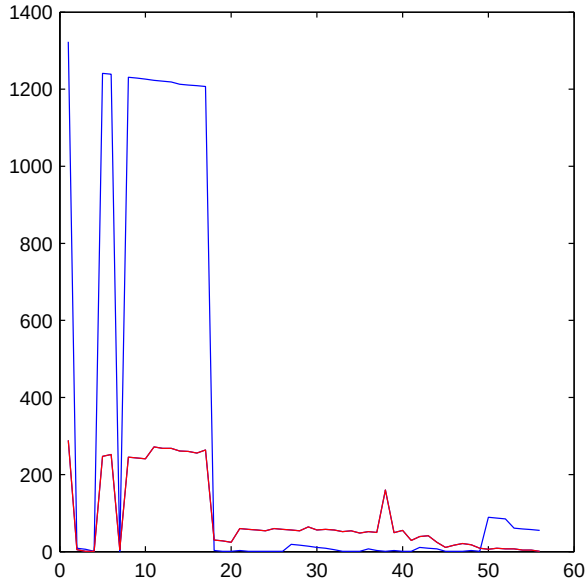


Figure 19: The blue curve and the red curve denote the number of vertex expansions for LPA* and m-LPA* respectively. The number on the x-axis is the index of the updated vertex along the initial shortest path. For LPA*, the number of vertex expansions is highest when the updates occur close to the source, while for m-LPA*, the use of multiscale information alleviates the computations during replanning when the update is close to the source (with slightly increased computational burden when the update happens in the largest d-square); the net gain in terms of worst-case computational complexity is clear from this figure.

1. The number of vertex expansions during replanning in LPA* varies dramatically from case to case. Specifically, when the updated vertex is closer to the source, or when the update generates a "local dead-end," a huge number of g-values may need to be recalculated.

2. The number of vertex expansions in m-LPA* is relatively insensitive with respect to the location of the updated vertices. The use of multiscale information reduces the number of vertex expansions when the blocking happens near the source; on the other hand, there will be a somewhat larger number of vertex

expansions than that of LPA* when the update happens in the largest d-square in the dynamically recursive dyadic partition tree, because in this case more vertices will be added to the beamlet graph during the replanning step.

These observations are illustrated in greater detail in Figures 20 and 21. Figure 20 shows two cases of replanning obtained from the LPA* algorithm, which exhibited widely different numbers of vertex expansions. The black cells are the source and destination. The gray cells indicate obstacles. The solid blue dots denote the cells that changed their status and are not traversable. The red circles identify the original path during the first step of LPA* (essentially A*), and the green stars indicate the updated shortest path obtained from the replanning part of LPA*. The yellow crosses denote the expanded vertices during replanning. In Figure 20(a), the blocking of a vertex in the gridworld induces a local "dead-end," and therefore all the g-values afterwards are recalculated. The updated shortest path deviates a few steps before the location of the blocking cell, before converging back to the original path afterwards. Because the blocking cell in Figure 20(a) is much closer to the source than that in Figure 20(b), the number of vertex expansions is also much higher, as expected.

Figure 21 shows two cases of m-LPA* replanning. In Figure 21(a) the blocking occurs near the source, as in Figure 20(a), but the number of vertex expansions is much smaller, since the usage of multiscale information provides a smaller size graph, and hence a smaller number of vertex expansions as well. Figure 21(b) illustrates the reason for an occasional increase in the number of vertex expansions in m-LPA*, especially when the vertex update occurs inside the largest d-square. In those cases, a larger number of free boundary cells is added to the beamlet graph as new vertices, following the recursive dyadic partitioning induced by the modified vertex. If the increased computational burden due to the newly-added vertices outperforms the gain from the multiscale structure of the beamlet graph, the number of vertex expansions can be higher than that of the LPA* implementation.

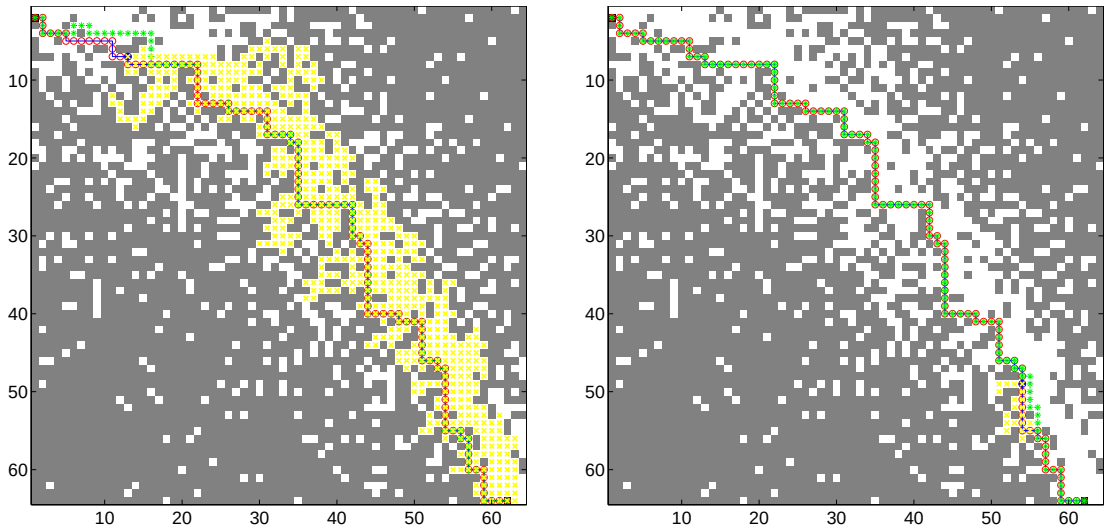


Figure 20: (a) Shortest path obtained from LPA* in the nearest neighbor graph with a large number of vertex expansions during replanning; (b) Shortest path obtained from LPA* in the nearest neighbor graph with a small number of vertex expansions.

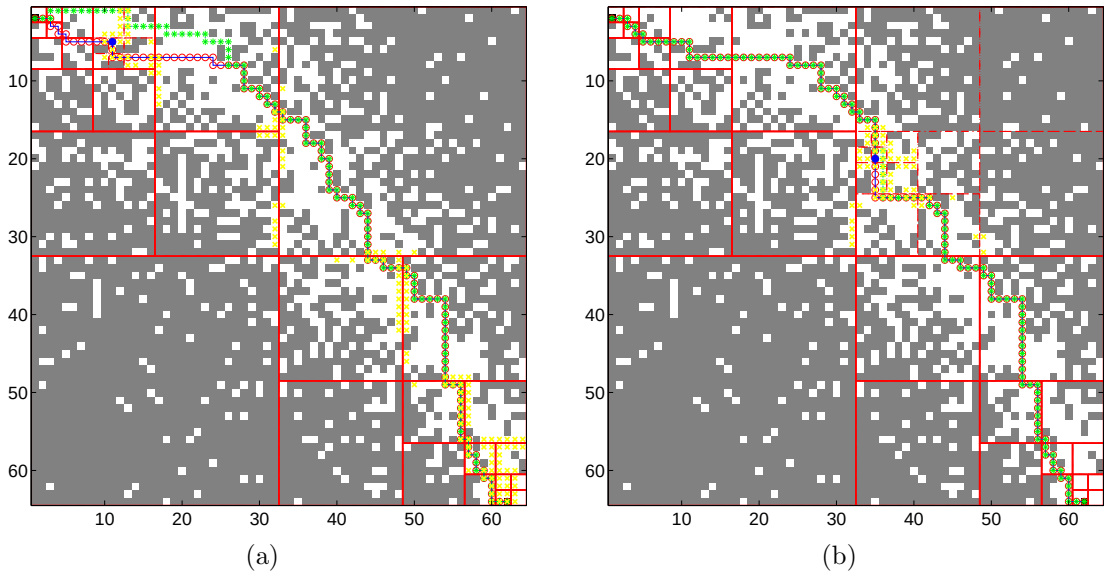


Figure 21: The shortest path-planning obtained from m-LPA* in the beamlet graph.

To further investigate the computational benefits obtained from the use of m-LPA*, we examined two more simulation scenarios as follows. We generated a grid-world similar to the previous case, but now we changed the parabolic curve to a circle. As before, comparisons were conducted using five randomly generated grid-worlds. Figure 22 shows the shortest paths obtained from the initial planning based on the nearest neighbor graph and the beamlet graph, respectively, while Fig. 23 shows two cases of replanning obtained from the LPA* and the m-LPA*, respectively.

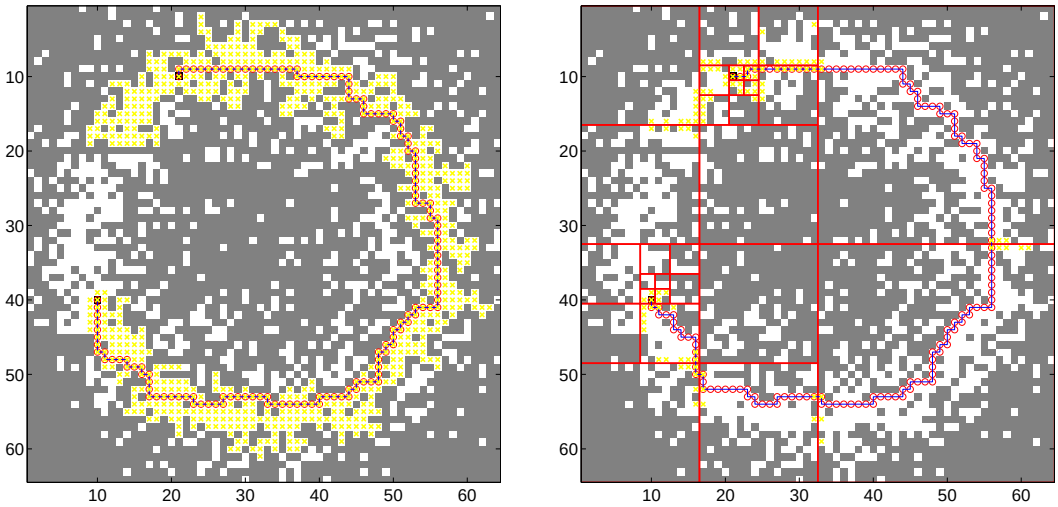


Figure 22: (a) The shortest path obtained from LPA* in the nearest neighbor graph. Notice that all free cells in the nearest neighbor graph are expanded. (b) The updated shortest path obtained from m-LPA* in the beamlet graph. The use of m-LPA* results in a much smaller number of vertex expansions in the initial planning step.

The third scenario involves a sinusoidal curve embedded in the environment. Figure 24 shows the initial planning obtained from the LPA* and the m-LPA*, and Figure 25 shows two cases of replanning based on the nearest neighbor graph and the beamlet graph, respectively. Figure 26 summarizes the results in terms of the number of vertex expansions for LPA* and m-LPA* for the last two scenarios. The number of vertex expansions for both LPA* and m-LPA* follows the same trend as in Figure 19. For LPA*, the number of vertex expansions is highest when the update happens close to the source, while for m-LPA* this number varies according to the

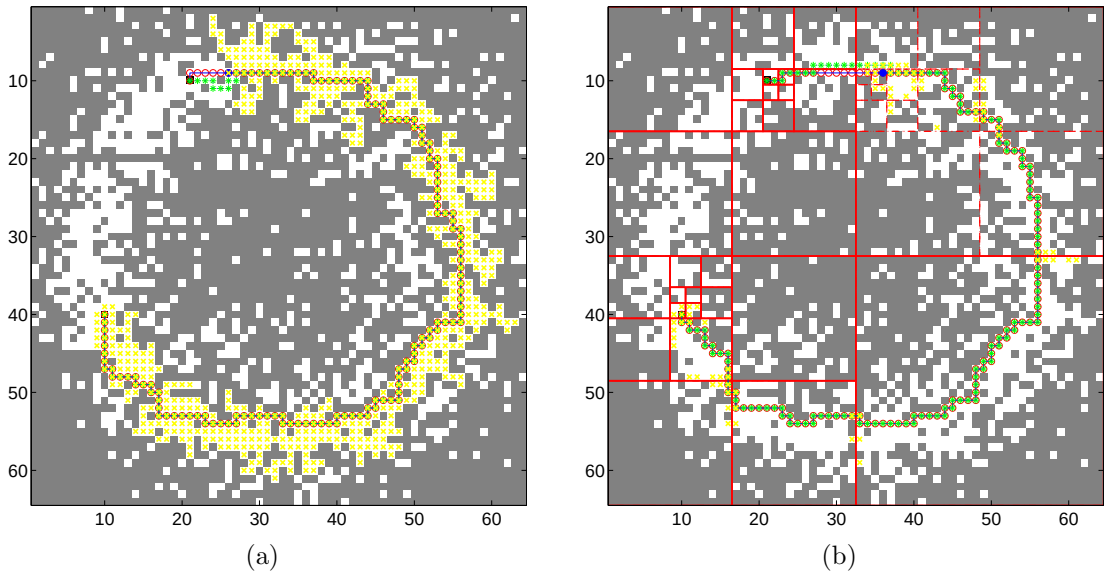


Figure 23: (a) The updated shortest path obtained from LPA* in the nearest neighbor graph. The blocking happens near the source, and hence all the g-values afterwards are recalculated. (b) The updated shortest path obtained from m-LPA* in the beamlet graph. The blocking happens in the upper-right coarsest scale d-square where further recursive dyadic partition takes place. Use of the multiscale information structure encoded in the DFPR-RDP significantly reduces the number of vertex expansions during replanning.

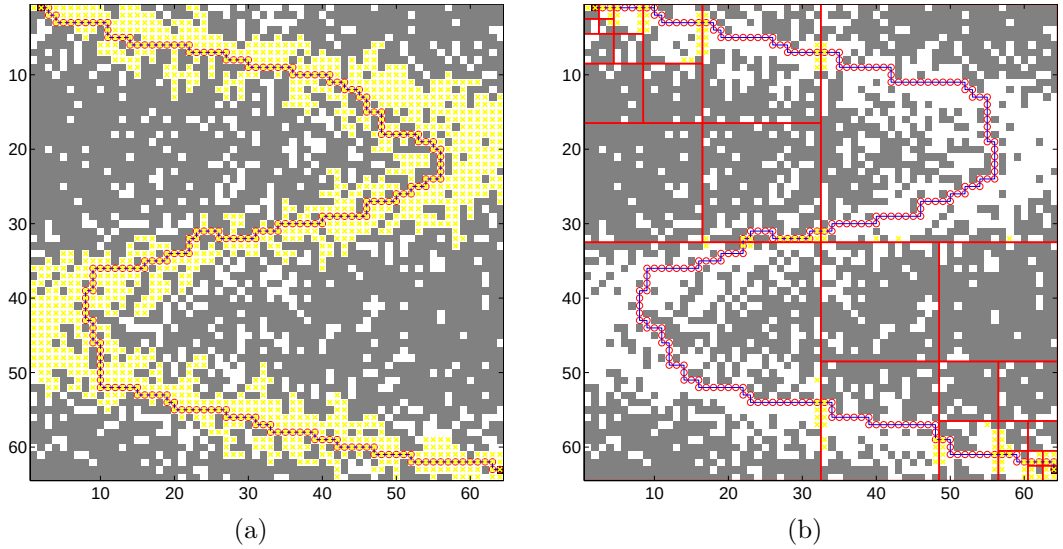


Figure 24: (a) The initial shortest path obtained from the LPA* in the nearest neighbor graph. Notice that all free cells in the nearest neighbor graph are expanded in the initial planning. (b) The initial shortest path obtained from the m-LPA* in the beamlet graph. The use of multiscale information greatly reduces the number of vertex expansions during the initial planning step.

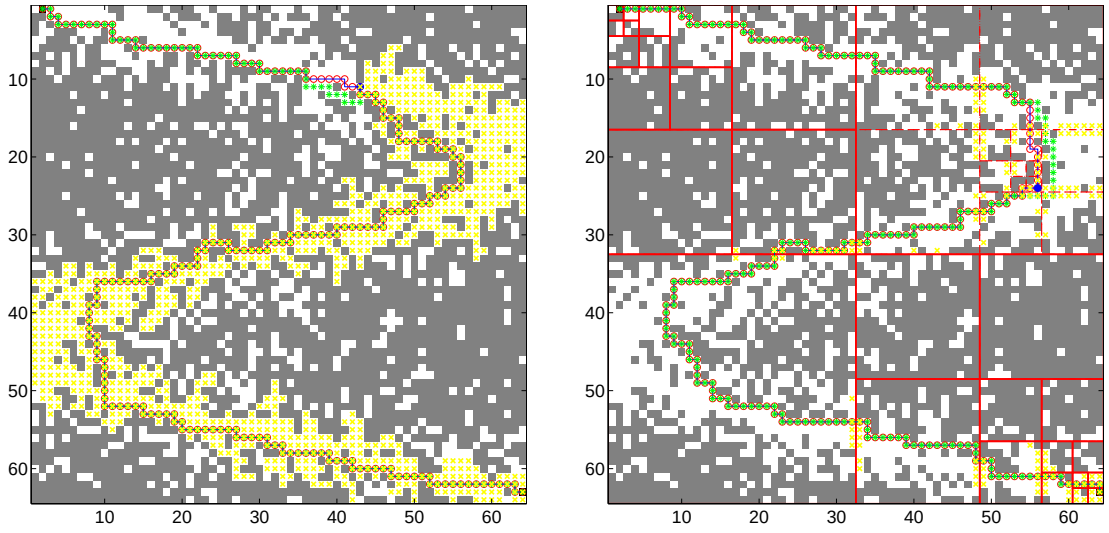
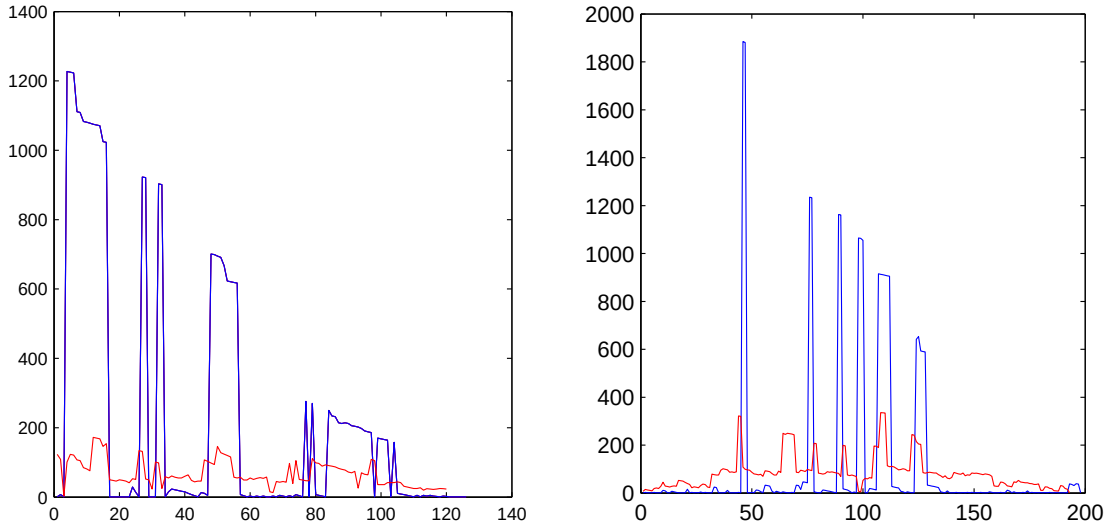


Figure 25: (a) The updated shortest path obtained from LPA* in the nearest neighbor graph. The blocking happens near the source and hence all the g-values afterwards are recalculated. (b) The updated shortest path obtained from m-LPA* in the beamlet graph. The blocking happens in the upper-right coarsest scale d-square where further recursive dyadic partition takes place. The use of multiscale information structure significantly reduces the number of vertex expansions during replanning.

closeness of the vertex updates to the source, and whether or not the updates are located in the larger d-squares.

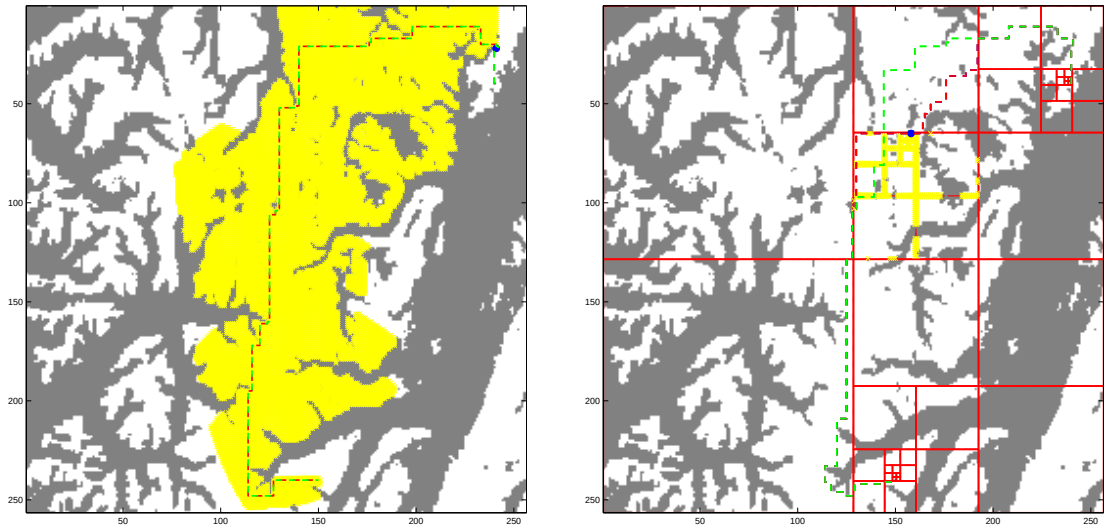
Finally, we reproduced a large-scale gridworld based on actual topographic data (i.e., elevation map) of a certain area in the US. Figure 27 shows two cases of replanning based on the nearest neighbor graph and the beamlet graph, respectively. The difference in the number of expanded vertices for the two cases, is clearly shown in Figure 27. Figure 28(a) provides a magnified version of Figure 27 (a) around the replanning area so that it can be seen clearly how the replanned path avoids the blocked vertex. Figure 28(b) summarizes the results in terms of the number of vertex expansions for LPA* and m-LPA* for this example, which illustrates the effectiveness of our proposed m-LPA* algorithm for large scale, realistic maps.

The results of the numerical simulations in this section confirm that m-LPA* is a more robust algorithm than the LPA* in terms of the number of vertex expansions



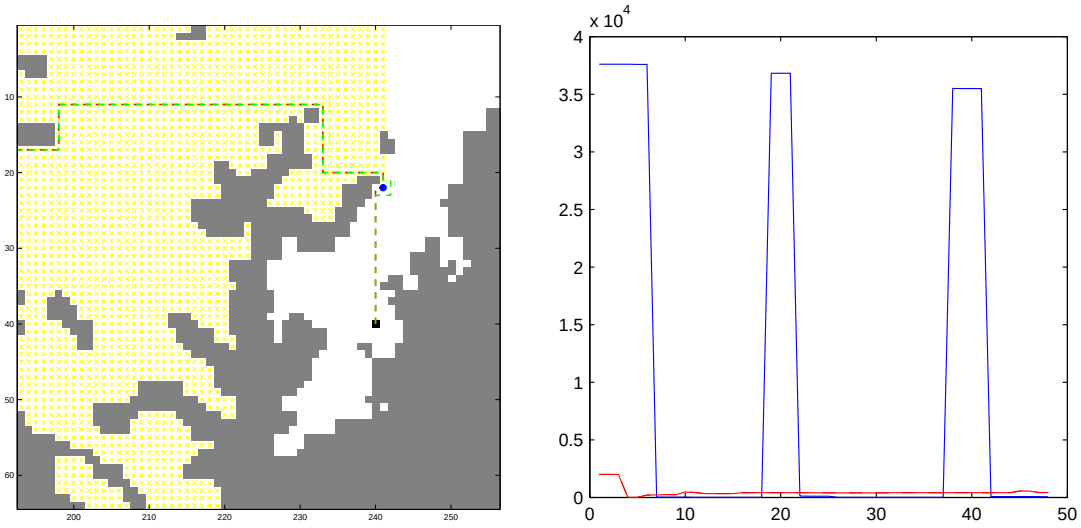
(a) Example environment of Fig. 22 and 23. (b) Example environment of Fig. 24 and 25.

Figure 26: Comparison of number of vertex expansions for both LPA* and m-LPA*. The blue curve and the red curve denote the number of vertex expansions for LPA* and m-LPA*, respectively.



(a) (b)

Figure 27: Comparison between LPA* and m-LPA* for a large map with real topographic data. Gray pixels indicate obstacles and white pixels are free. (a) The blocking happens near the source and hence all the g-values afterwards are recalculated. (b) The updated shortest path obtained from m-LPA* in the beamlet graph.



(a) Magnified Area in Fig. 27 (b) Example of Fig. 27

Figure 28: (a) Magnified replanning area of the simulation shows the correctness of the m-LPA*; (b) Comparison of vertex expansions for both LPA* and m-LPA*. The blue curve and the red curve denote the number of vertex expansions for these two methods, respectively.

during replanning for a large variety of test scenarios. It is observed that the number of vertex expansions in m-LPA* does not vary dramatically with the location of the updated vertex whose traversability properties change. This mitigation of the volatility in the number of vertex expansions achieved by m-LPA* can be a desirable property in many applications.

3.7 Discussion and Related Prior Work

The proposed m-LPA* algorithm provides a novel, non-trivial extension in the family of incremental search algorithms operating on quadtree-based data structures. Quadtrees are, in fact, a widely adopted hierarchical representation used to encode obstacles in a given 2D image/environment. They partition an image by recursively subdividing it into four smaller quadrants, and these successive subdivisions continue until either a subregion which is free of obstacles is found, or the finest resolution is reached. Because of their efficient memory requirements, quadtrees have become popular in path-planning applications.

Several previous path-planning algorithms exploit the nice properties of quadtrees. One of the earlier approaches is given in [63]. This work proved that a quadtree structure is better than the use of regular grids to represent 2D environments. The benefits of quadtrees stem from the fact that they allow for an efficient partitioning of the environment so that single cells can be used to encode large empty regions. Quadtrees also reduce the memory requirements since they use a smaller number of cells. In early path-planning implementations, first a quadtree is created to represent the 2D workspace, and then the path is generated by joining the line segments between the centers of the cells. However, this method merely finds a suboptimal solution. To remedy this shortcoming, the "framed quadtree" data structure was introduced in [76]. In a framed quadtree, free cells augmented the perimeter of each quadrant; these free cells are then used to pass through the different quadtree regions. The optimal path is generated by combining these free cells with line segments. At first glance, the framed quadtree data structure resembles the beamlet graph structure in this chapter. However, [76] does not introduce a beamlet-like connectivity to explore the finer scale information contained in the quadtree decomposition of the environment. Consequently, no "fusion" algorithm is used to efficiently organize the pre-computed information across all levels.

Another recent methodology that is somewhat similar to our work is the hierarchical path finding A* (HPA*) algorithm of [6]. This is a popular algorithm for real-time path-finding problems, especially for video game applications. The key idea of HPA* is to divide the map into clusters (corresponding to equally-sized squares) and generate a new graph (the so-called abstract graph) by using the information of the free cells belonging to the boundary of these clusters. The motivation behind such an approach is that for many real-time path-finding applications, the complete path is not needed. Knowing the first few moves of a valid path often suffices, allowing a mobile agent to start moving in the right direction, even before the whole path has

been computed. Using this point of view, the authors of [6] derived a clustering algorithm that groups–when appropriate–neighboring nodes together to pre-process the grid and build a higher level graph (repeatedly if necessary) at multiple levels. The suboptimal path is found by searching at the top level first and then via recursively planning more detailed paths at the lower levels. Cached solutions of paths are often reused in the hierarchy, so a tradeoff can be made between memory and computation. Enhancements of the original HPA* are provided in [43].

At first glance, HPA* bears a strong resemblance with m-LPA*, in the sense that they both construct a hierarchical decomposition of the environment. However, beamlet-like connectivity is absent in [6, 43] and the vertices of the abstract graph are just a subset of the original gridded map. Furthermore, at least two more significant differences exist compared with our work: first, the HPA* seeks an approximate solution fast, while the m-LPA* (and the m-A*) will find the exact, optimal solution to the shortest path problem. As mentioned above, HPA* was designed with computer games in mind, where only the initial few steps of the shortest path matters. This explains why the inexact solution is not a concern in HPA*. Instead, m-LPA* is designed when the exact shortest path needs to be found. Second, there are significant differences in the algorithmic design as well. For example, m-LPA* refines the grid-world into the finest resolution near the source and destination, whereas HPA* does not. Handling many scales in HPA* becomes difficult. The numerical examples in [6] use at most three levels. On the other hand, m-LPA* uses $\log n$ scales. The m-LPA* adopts a "bottom-up" fusion algorithm to organize pre-computed information. This is not present in HPA* either.

3.8 Conclusion

In this chapter, we have presented a novel extension of the well-known Lifelong Planning A* (LPA*) algorithm based on a multiscale decomposition of the environment.

Our algorithm may be viewed as an extension of both the classical LPA* algorithm and the recently proposed multiscale A* (m-A*) algorithm. The bottom-up fusion algorithm of m-A* is used as a multiscale strategy to preprocess the information at multiple scales and construct the beamlet graph. The proposed multiscale LPA* algorithm (m-LPA*) provides a significant reduction in terms of vertex expansions over the original LPA* algorithm at the worst case. Since the implementation of LPA* is based on a Fibonacci heap data structure for maintaining the priority queue, the vertex re-expansion dominates computations, which means that a significant reduction in terms of vertex re-expansions will result in a significant reduction in terms of computational time. These insights are confirmed by our numerical examples. Extensions of the proposed multiscale algorithm that adapts to a moving source, as well as extensions to higher dimensions are possible, and are currently under investigation.

CHAPTER IV

SPOT ELECTRICITY SPIKE PREDICTION VIA COMBINATION OF BOOSTING TREES AND WAVELET ANALYSIS

4.1 Introduction

Electricity spot price modeling is a challenging yet very important task for both electricity market participants and the system operators. Due to the large-scale system complexity and a wide variety of uncertainties in the power grid, electricity prices are highly volatile and their volatility is routinely exacerbated by the price spikes. It is not uncommon for the magnitude of the price jumps to be tens or even hundreds of times higher than that of the normal prices. This attests to the need for accurate price-spike forecast models in managing price risk exposures.

To model electricity price dynamics and capture their characteristics, one strand of research on modeling electricity price processes focuses on the aspect of derivative pricing and asset valuation, which investigates electricity spot and forward price models in a risk-neutral framework. Another research strand concerns the modeling of electricity prices in the physical world, and offers the price forecasting to assist physical trading and operational decision-making. In this chapter, we focus on the second strand. A new method, which combines the strength of nondecimated wavelet analysis and the powerful boosting algorithm in machine learning, is proposed to model the electricity spot price. The major contribution of the proposed methodology is three-fold. First, it offers a systematic approach for detecting and predicting price spikes in the real-time electricity markets as opposed to the less volatile day-ahead markets. Second, the boosting decision-tree framework enables the analysis on the

relative importance of the fundamental supply and demand factors in predicting price spikes. Such information is essential to the design of price-spike mitigation mechanisms. Third, this new methodology significantly increases the prediction accuracy. The case studies conducted in the two major regional markets in Australia illustrates the universal applicability of the new model.

There are many proposed models in the area of electricity price modeling. These include time series models (e.g., the ARIMA model proposed in [10]) and Neural Networks models (in [64, 35]). The *Support Vector Machine model (SVM)* is introduced in [77] for the classification of spikes in electricity spot prices in the Australia electricity market. Another model is proposed in [55], in which the regime switching model with time-varying parameters is proposed for spot price modeling in the U.S. electricity market. While only the day-ahead price is considered in their modeling, our proposed approach focuses on quantitative characterization of spikes in real-time electricity spot prices, which is well recognized as a more difficult task. Moreover, the identification of a spot price spike is rather arbitrary in the existing models and we propose a systematic approach to identify spikes. In Section 4.4, it is shown that the proposed methodology outperforms the existing methods significantly.

The proposed modeling framework consists of two modules: spike detection by *Nondecimated Wavelet Transform* and spike prediction by *Boosting Trees*. One prominent difference between the electricity price and other commodities is the daily seasonality. One novel approach to address this issue is to model the hourly prices within a day as one 24-dimensional vector as in [9]. Basically the univariate time series modeling problem is converted into a multivariate data modeling problem, which represents the intraday seasonality in time series through the cross-sectional relationship among different dimensions. The seasonality can therefore be explicitly modeled and the high frequency data is converted to the low frequency data. Both facilitate the detection of spikes in electricity spot prices.

The nondecimated wavelet transform is a key step in the price spikes detection module. With this transform, the wavelet coefficients are invariant to the origin of the time series, which is a desirable property for time series analysis. Meanwhile, the length of time series in nondecimated wavelet analysis can be arbitrary, i.e., not necessarily being a power of 2. Basic idea for price spike detection is as follows. The scaling coefficients from nondecimated wavelet analysis provide an approximation of the trend, which incorporates the information of weekly and monthly seasonality. After removing this trend from spot electricity prices, a more precise definition of spikes (in comparison to the traditional hard-thresholding definition) is obtained. In price spike prediction module, several factors, such as supply/demand, net interchange and clustering effect within the occurrence of spikes, are considered as predictors in the boosting trees. This methodology inherits the nice properties of decision trees, which includes the robustness to irrelevant covariates and adaptiveness to mixed type of random variables. Boosting trees is an ensemble method and is usually more efficient computationally than SVM in the sense that it requires less tuning parameters to be determined by cross validation. In addition the relative importance of features can be calculated and interpreted as a practical guidelines for feature selection.

In the rest of this chapter, Section 4.2 discusses the market structure of deregulated electricity market and the data set used. Section 4.3 contains the detailed description of each module in our model. Section 4.4 presents case studies with electricity market data of both Queenland and New South Wales in Australia. Discussion of computational efficiency and summary statistics are given in Section 4.5. Section 4.7 concludes and discusses future extensions.

4.2 Background and Dataset

The electricity market we consider in this paper is the NEM of Australia so that we can compare the prediction accuracy with methods reported in existing literature.

The NEM of Australia began operating as a wholesale market for supply of electricity to retailers and end-users in Queensland, New South Wales, Victoria and South Australia in Dec. 1998. The power system that supports the *NEM*, by geographic area, is the *largest* interconnected electricity network in the world. Exchange between electricity producers and consumers of electricity are facilitated through a pool where the output from generators is aggregated and scheduled to meet demand. Wholesale trading in electricity is conducted as a spot market where supply and demand are instantaneously matched in real-time through a centrally-coordinated dispatch process. Generators offer to supply the market with specific amounts of electricity at particular prices. Offers are submitted every five minutes of a day. From all offers submitted, AEMO chooses the generators and determines the amount of electricity they produce based on the principle of meeting prevailing demand in the most cost-efficient way. AEMO then dispatches these generators into production.

The electricity prices from both Queensland and New South Wales are analyzed in case studies in Section 4.4. Our model is based on the aggregated hourly spot price. The behaviour of the spot electricity price is usually decomposed into two components: long-term trends and spikes. Long-term trend incorporates the information about trend and seasonality, while the price spikes contain the information about either short-term imbalance between supply and demand or other rare events. To model the long-term trend, the methods adopted in the literature are time-series based methods such as ARIMA; on the other hand, the occurrence and range of spikes are usually predicted by classification algorithms.

A price spike is an abnormal market clearing price at a time point t, which is significantly different from the price at previous time point $t-1$. Price spikes have a significant impact on the profit of market participants, especially those who are on hedge contracts. Price spikes may last for a few time-units and are highly stochastic. The abnormal price spikes can be classified into three categories: (1) the price that is

much higher than the normal price so that it is named as *abnormally high price*; (2) the difference between two neighboring prices is larger than a threshold, which leads to the *abnormally jump price*; (3) price that are smaller than zero, which is classified as *negative price*. In this paper, we will focus on the prediction of the *abnormally high prices*.

The definition of *abnormally high price spike* should be carefully examined. In [71], it provides four common methods to define the price spikes: (1) hard-thresholding method; (2) upper quantile method; (3) moving average detrending method; (4) wavelet method. Among these four methods, the moving-average and wavelet methods take into account the trend and seasonality in spot electricity prices. We employ the "wavelet method" with modification and compared it to the hard-thresholding method. We consider the wavelet-based definition as a reasonable alternative scheme of price spikes identification. The details of our methodology are provided in Section 4.3.

4.3 Methodology

In this section, the details of the proposed methodology are discussed. The flow chart of our model is provided in Fig 29. to give a high-level profile of the newly proposed method. A review of wavelet analysis is presented in Section 4.3.1.1. Section 4.3.1.2 presents a conceptually simpler definition of spot price spikes, which is more reasonable alternative than the previous ones in the literature. Section 4.3.2 provides a brief introduction to the boosting method and illustrates its advantages over other classification methodologies in spot price spike prediction. The feature selection for the model is given in Section 4.3.3. Section 4.3.4 contains several criteria that will be utilized for comparison of prediction capability of different models.

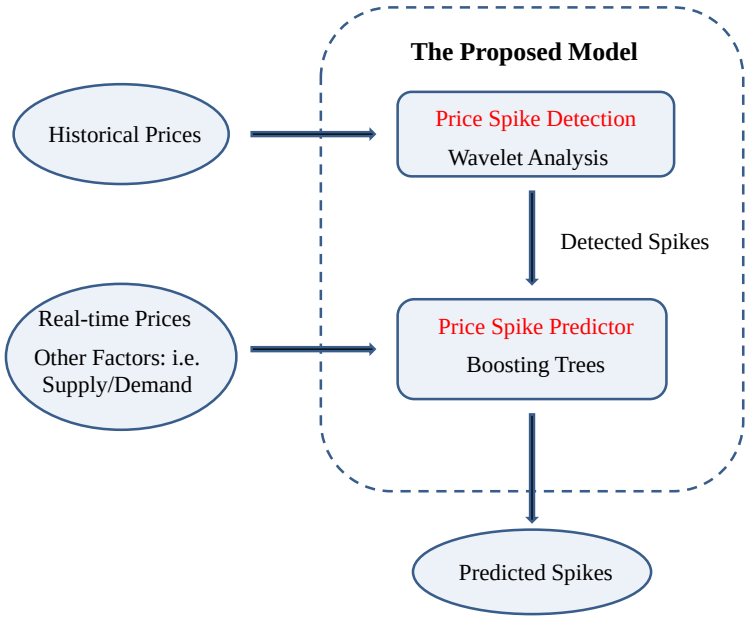


Figure 29: A flow chart of modules in our proposed method. Inputs are the historical/real-time prices and predictors such as demand and supply. The two modules are highlighted in red.

4.3.1 Detection of Electricity Price Spikes

In this section, we will describe our approach for spikes detection in hourly electricity spot prices. First we review *nondecimated wavelet analysis* in Section 4.3.1.1 as the mathematical preparation and then we will specify our procedure for spikes detection in Section 4.3.1.2.

4.3.1.1 Review of Wavelet Analysis

The wavelet transform involves the projection of a signal onto a set of components—the so-called wavelets. Unlike sines and cosines, individual wavelet functions are localized in time and space simultaneously [58]. Wavelets belong to families, like the Daubechies wavelet family used in our setting (which is a common choice). A wavelet family comes in pairs of father (S) and mother wavelets (D). The former represents the "lowest frequency" smooth components—those requiring wavelets with

the widest support—whereas the latter captures the "higher frequency" detail components. Mathematically, given the signal $p(t)$ which is a continuous function, its (decimated) wavelet decomposition has a generic form

$$p(t) = \sum_{k \in \Lambda_J} \alpha_{J,k} \phi_{J,k}(t) + \sum_{j>J} \sum_{k \in \Lambda_j} \beta_{j,k} \psi_{j,k}(t)$$

where $\phi_{J,k}(t)$ and $\psi_{j,k}(t)$ are scaling functions at the coarsest scale J and wavelet functions respectively. Generally,

$$\alpha_{j,k} = \int p(t)\phi_{j,k}(t)dt = \int p(t)\phi(2^j t - k)dt$$

$$\beta_{j,k} = \int p(t)\psi_{j,k}(t)dt = \int p(t)\psi(2^j t - k)dt$$

which can be seen as the projections of signal onto the orthogonal basis in Hilbert space. In practice, a nice feature of wavelet decomposition is that it has fast discrete algorithms. Decimated Discrete Wavelet Transform (DWT) is just a linear transform of the original discretely sampled signal. More specifically, wavelet coefficients $W = \mathcal{W}P$, where P is the equally sampled signal and $\mathcal{W}$ is the orthogonal linear transform. Here W is a vector that contains all scales of wavelet coefficients and the coarsest scaling coefficients. *Nondecimated discrete wavelet transform* is a modified DWT, in which the scaling and wavelet coefficients for each scale are of the same length as the original signal, and also inherits the properties of *Multiresolution Analysis* and *ANOVA* from DWT. In fact, the coefficients of nondecimated wavelets are obtained by passing the signal through renormalized DWT filters and the multiresolution decomposition are defined in a similar way. The mathematical details are discussed in [58]. Besides, the nondecimated wavelet analysis is defined for all samples size (i.e. the length needs not to be a multiple of power of 2). We refer to [58, 20] for more applications of wavelets in time series analysis.

4.3.1.2 Procedures of Spikes Detection

The following steps are needed for the detection of spikes in each hour:

(i). log-transformation of hourly prices.

(ii). wavelet decomposition of time series for each hour.

(iii). detection of spikes on the original scale.

We first conduct the following transformation to the spot price

$$\tilde{P}_t = \text{sign}(P_t) \log\left(|P_t| + 1\right)$$

where P_t and $\tilde{P}_t$ are the raw and transformed series of electricity spot prices. The logarithmic transformation could mitigate the influence of outlier (i.e., huge price spikes) in the wavelet analysis. The *sign* and absolute values deal with negative spot prices. This is more accurate than replacing the negative price with moving average.

For wavelet decomposition, we introduce two indices into the price series $\tilde{P}_i^h$, where i indicates the day and h indicates at which hour of day i the price is taken. We assume that the first hour of a day starts at 4:05 A.M. So P_i^1 indicates the aggregated price between 4:05 A.M. and 5:00 A.M. on ith day. P_i^2 indicates the aggregated price between 5:05 A.M. and 6:00 A.M. on the ith day, and so on. The double indexed price series can be divided into 24 time series as follows: $\{P_i^h : i = 1, 2, ..., N\}$ for $i = 1, 2, ..., 24$, where N is the number of days in our data set. For fixed i, we apply the nondecimated wavelet to decompose it into two components: $\{\tilde{P}_i^h\} = \sum_{j=1}^{J} \mathcal{D}_{ij}^h + \mathcal{S}_{iJ}^h$, where j denotes the level in multiresolution analysis. Then the detrended series are simply $\mathcal{S}_{iJ}^h$ for day i and hour h. The 24 hour-based time series are then combined into one time series according to $\hat{P}_{24i+h} = \text{sign}(\mathcal{S}_{iJ}^h)\left(\exp|\mathcal{S}_{iJ}^h| - 1\right)$. Apparently, $\{\hat{P}_t, t = 1, 2, ..., 24 \times N\}$ is a univariate time series, which will be our approximation of trend in spot electricity prices in remaining of the paper.

In our model, difference between the original series and the approximation of trend is considered as the residual, denoted as $r_t = P_t - \hat{P}_t$. The elements in the residual which are larger than K times median absolute deviation (MAD) in each hour, i.e. $|r_t| > K\text{mad}(r_t)$, are defined as spikes (K ranges from 3 to 10 depending on the data). The median absolute deviation is used instead of standard deviation due to

the consideration of nonnormality. The value of K is chosen so that the proportion of spikes detected will be similar to that from hard-thresholding method. Fig. 30 illustrates the difference between the two definitions for electricity spikes. It is shown that some spikes that are dominated by the trend and seasonality will be missed under hard-thresholding method.

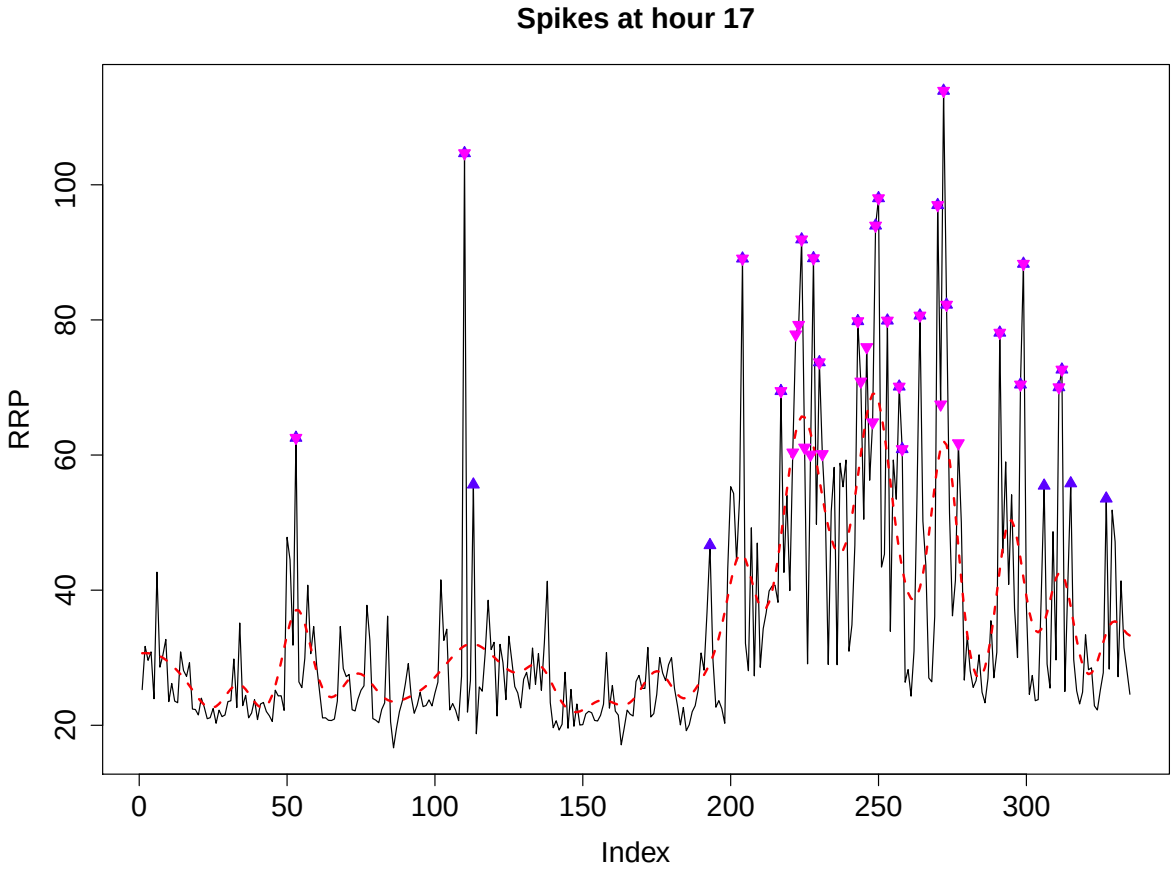


Figure 30: Queensland Electricity Markets. Residuals that are 3 median absolute deviation away from the trend, which is approximated by smooth component in wavelets analysis, are defined to be spikes. For hard-thresholding, prices which are larger than 60 are defined to be spikes. The black curve shows the spot electricity price and the dashed red line represents the trend identified by multi-resolution analysis. The blue triangle point-up and pink triangle point-down denote the spikes identified by wavelet-based and hard-thresholding based definition respectively.

4.3.2 Prediction Model: Boosting Trees

In this section, we will describe the ideas of gradient boosting machine proposed in [28, 29]. Boosting is one of the most powerful learning algorithms introduced in the last ten years. The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers to produce a powerful "committee." A weak classifier is the one whose error rate is only slightly better than random guessing. The boosting will sequentially apply the weak classification algorithm ("basic learner") to repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction. The essential task in each iteration is to adjust the weight to each sample and thereby force each successive classifier to concentrate on those training observations that are missed by previous ones in the sequence.

The weak classifier used in our model is the decision tree due to its attractive properties which are not possessed by support vector machine. The intuition of single decision tree is quite simple: it just conducts the recursive partition of the state space generated by the covariates, and approximates the true regression function by piecewise constant function that relies on the partition. The desirable properties of decision tree include the capability of handling mixed type of data, missing values, its robustness to outliers in state space, its computational scalability and the ability to deal with irrelevant covariates. Clearly one single classification tree does not have strong predictive power. The boosting of decision trees therefore enhances the predictive capability significantly while preserving the good properties referred to earlier.

AdaBoost proposed in [27] is considered as a classical boosting method and is equivalent to forward stagewise additive logistic regression aimed with exponential loss function. Like the hinge loss function in Support Vector Machine, the exponential loss function is also an upper bound of classification error. Because it is

continuous and differentiable, this makes the numerical computation feasible. Most importantly, minimization of the upper bound of classification error avoids the over-fitting effectively.

In our model, we employ *Gradient Boosting* method proposed in [29] which is a generalization of Adaboost. The objective is to find a regression function $\hat{f}(\mathbf{x})$ that minimizes the conditional expectation of some loss function $L(y, f)$, i.e.

$$\hat{f}(\mathbf{x}) = \text{argmin}_{f(\mathbf{x})} E_{y|\mathbf{x}}[L(y, f(\mathbf{x}))|\mathbf{x}] \tag{50}$$

where $\mathbf{x}$ is a vector containing features related to electricity spikes that will be discussed in Section 4.3.3. To estimate the function nonparametrically, the intuition is to modify the current estimate of $f(\mathbf{x})$ by adding a new function to $f(\mathbf{x})$ in a greedy fashion. More specifically, Let $f_i = f(\mathbf{x}_i)$, then decrease the approximated loss function

$$J(f) = \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) = \sum_{i=1}^{N} L(y_i, f_i) \tag{51}$$

where N is the sample size. The negative gradient of $J(f)$ indicates the direction of the locally greatest decrease in $J(f)$. This tells us to modify f as

$$\hat{f} = \hat{f} - \rho \nabla J(f) \tag{52}$$

where ρ is the size of the step along the direction of greatest descent. The idea of gradient boosting is thus to select the class of functions that use the covariate information to approximate the gradient, using a regression tree. At each iteration the algorithm determines the direction, the gradient, in which it needs to improve the fit to the data and selects a particular model from the allowable class of functions that is in most agreement with the direction. Here the values f_i are implicitly treated as the parameters in the optimization problem and the process of seeking descent gradient is equivalent to the seeking of appropriate weak classifier iteratively.

The final function $\hat{f}$ is essentially an additive model of weak learners. It is well

known [28] that $\hat{f}$ is the estimator of

$$f^*(\mathbf{x}) = \text{argmin}_{f(\mathbf{x})}\mathbb{E}_{Y|\mathbf{x}}(e^{-Yf(\mathbf{x})}) = \frac{1}{2}\log\frac{\Pr(Y=1|\mathbf{x})}{\Pr(Y=-1|\mathbf{x})}.$$

Therefore, $\hat{p}(\mathbf{x}) = \frac{e^{\hat{f}(\mathbf{x})}}{e^{-\hat{f}(\mathbf{x})}+e^{\hat{f}(\mathbf{x})}}$ and the classification $\hat{y}_i = \text{sign}(\hat{f}(\mathbf{x}_i))$ is the natural choice for classification.

Notice that the whole process contains two steps of approximation: using regression trees to approximate the gradient and using the exponential loss function to approximate the classification error.

4.3.3 Selected Features for Prediction

Feature selection is to choose the attributes relevant to the problem. Based on the statistical analysis the following attributes are chosen for the classifier: boosting trees.

- *Demand*. It is well known that the demand is highly relevant to the market cleaning price in most electricity markets. As the demand increases, the probability of price being a spike becomes higher.

- *Supply*. Supply and demand usually determine the occurrence of price spikes. [34] provides an excellent introduction of the roles played by supply/demand in the price-spike formation.

- *Existence*. This feature describes the relationship between a spike and its predecessors. It can only take two values: 0 or 1. Existence at a time point $t > 0$ is defined as:

$$I_{\text{ex}}(t) = \begin{cases} 1 & \text{if there are spikes in } (\lfloor t/24 \rfloor \times 24, t) \\ 0 & \text{if } n \text{ is odd} \end{cases}$$

  where N is a counting measure, which characterizes the arrivals of spikes. This feature is included since it captures the clustering pattern of spikes which are

usually caused by some short-term events, such as contingencies, market practice and transmission congestion. These short-term events do not usually happen and last for a long time.

- *Time within the day.* The likelihood of observing spot price spikes shall be higher during the peak hour in each day. Therefore this discrete covariate would contribute to the prediction of spikes.

- *Net interchange and dispatchable load.* Dispatchable loads are the electricity load that are registered to participate in the central dispatch and pricing process with their capability in promptly reducing the consumption quantity.

4.3.4 Performance Measure

Three measures are adopted to assess the results from case studies. For a general classification problem, the most basic performance measure is the *confusion matrix* consists of the true positive rate, true negative rate, false positive rate and false negative rate. The naive classification error is defined as:

$$\text{classification error} = \frac{1}{n}\sum_{i=1}^{n} 1_{\tau_i \neq \hat{\tau}_i}$$

where τ and $\hat{\tau}$ denote the true and estimated locations of spikes. However, in the electricity spike detection problem, it is not appropriate in the sense that the data are significantly imbalanced. With only a small proportion of prices identified as spikes, the classification error would be very lower even if all the spikes are misclassified. Therefore the *true positive rate* and *false positive rate* are more relevant to our spike prediction problem and hence studied in the numerical experiments.

Receiver Operator Characteristics (ROC) curve is a popular tool in both signal detection theory and the evaluation of binary classifier. It is a graphical plot of the sensitivity or true positive rate versus false positive rate as its discrimination threshold is varied. One good review of its application in machine learning is provided in [7].

In our numerical experiment, ROC curves from boosting trees and SVM are drawn to show the difference between their predictive power. All our numerical comparisons in Section 4.4 are based on these three criteria.

4.4 Case Study

In this section, we will provide the numerical study of our model on the real-world NEM electricity spot price. More specifically, the data period is from Oct. 2009 to Oct. 2010. To predict electricity spikes in the next hour, input features in both boosting trees and support vector machine are current demand, supply, existence, time within the day and dispatchable load. Recall that mathematically, conditional probability of having a spike at $t+1$ in boosting trees is as follows.

$$\Pr(\text{spike}_{t+1} = 1|\mathbf{x}_t) = \frac{1}{1+\exp(-2\hat{f}(\mathbf{x}_t))}$$

where vector $\mathbf{x}_t$ contains the features mentioned in Section. 4.3.3 and $\hat{f}$ is learned from boosting trees. Different performance measurement are included for our model evaluation.

4.4.1 Price Spikes Prediction in Queensland

In the proposed method, we set the maximum number of decision trees used in boosting to be 4000, and choose the coarsest scale $J = 3$ in nondecimated wavelet decomposition. We also examine the case of $J = 5$ and the results are the same qualitatively and hence are omitted. The choice of K, as long as it is in a reasonable range, does not affect the results of classification. 33% of data (i.e., training sample size of 2680) is utilized in the training and the remaining is used for testing purpose. 5-fold cross-validation is used to determine the tuning parameter in boosting: the number of decision trees.

The results of model training are summarized in Fig. 31, which shows the performance of our proposed method for spike prediction. Notice that the values of

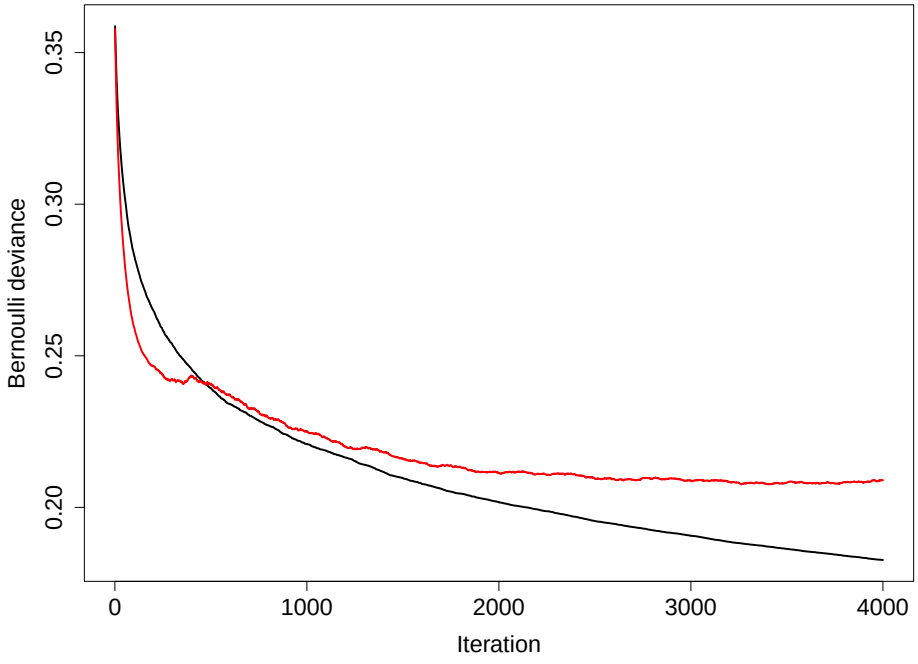


Figure 31: Summary of results from boosting trees with wavelets. x-axis records the number of boosting trees, which indicates the model complexity. On y-axis is shown the value of empirical loss. The black curve shows the training error. The red curve shows the upper bound of prediction error of cross-validation.

loss function in boosting is shown in Fig. 31 versus number of trees (i.e. complexity of boosting). We also report the performance comparison based on our previously defined measures in Table 4. The first part of Table 4 shows the comparison relied on the wavelet-based definition, while the second part shows the results from hard-thresholding method. For both boosting trees and SVM, we report the true positive rate and false positive rate with respect to both spike definitions to illustrate the consistency of our comparison. In wavelet-based definition, cases where $K = 6, 8$ are reported, while in hard-thresholding cases, threshold $75, 85$ are considered.

The value of τ is the threshold for conditional probability. Different values of τ are employed to delivery similar true positive rate or false alarm rate so that the prediction capability could be evaluated clearly. Among almost all comparisons, boosting tree dominates SVM in the sense that it has higher true positive rate and lower false alarm rate. The user can choose different threshold values to achieve trade-off between true postive rate and false alarm rate based on their risk aversion.

Table 4: Summary of Numberical Experiments: QLD

Wavelet	6σ		8σ	
	Boosting	SVM	Boosting	SVM
Tpr	50.10%	42.30%	54.29%	54.29%
Fpr	1.24%	1.47%	0.49%	0.64%
τ	0.4	0.4	0.6	0.2
Hard	75		85	
	Boosting	SVM	Boosting	SVM
Tpr	54.20%	54.20%	43.70%	43.75%
Fpr	1.67%	1.69%	0.18%	0.17%
τ	0.4	0.08	0.4	0.4

The best iteration numbers in boosting trees and parameters in SVM are selected by 5-fold cross validation. Other kernels for SVM have also been tested in our case study, and we only report the results from Gaussian kernel because it delivers the best

prediction accuracy. From all these results shown in Table 4, boosting trees stand out as a superior binary classifier compared to SVM. Fig. 32 shows the performance of boosting trees with hard-thresholding detection of spikes on training and cross-validation data set. Both Fig. 31 and Fig. 32 shows the capability of boosting trees to avoid overfitting and its stability in its tuning parameter.

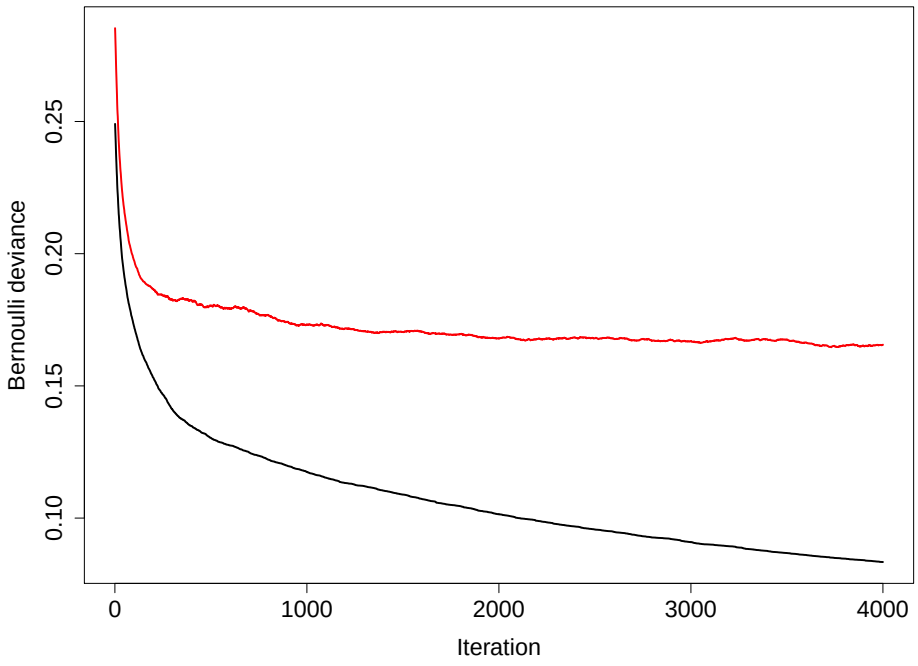


Figure 32: Summary of results from hard-threshold boosting trees. The black curve shows the training error. The red curve shows the upper bound of prediction error on cross-validation.

A more extensive comparison between the predictability between boosting trees and support vector machine is achieved by examining *Receiver Operator Characteristics (ROC)* curve. Fig. 33 represents ROC curves from both boosting trees and SVM based on wavelet-based detection of spikes. Area under curve (AUC), which is quantity that is proportional to the predictability of classifier, is also shown. The better performance of boosting trees over SVM can be visualized clearly in Fig. 33. The ROC curves of hard-thresholding based classifiers looks very similiar to Fig. 33

and are omitted. Another merit of ROC is that the curve shows cleary the trade-off between the true positive rate and false alarm rate. The practitioner can choose the threshold for conditional probability to achieve the best performance which is consistent with their risk tolerance.

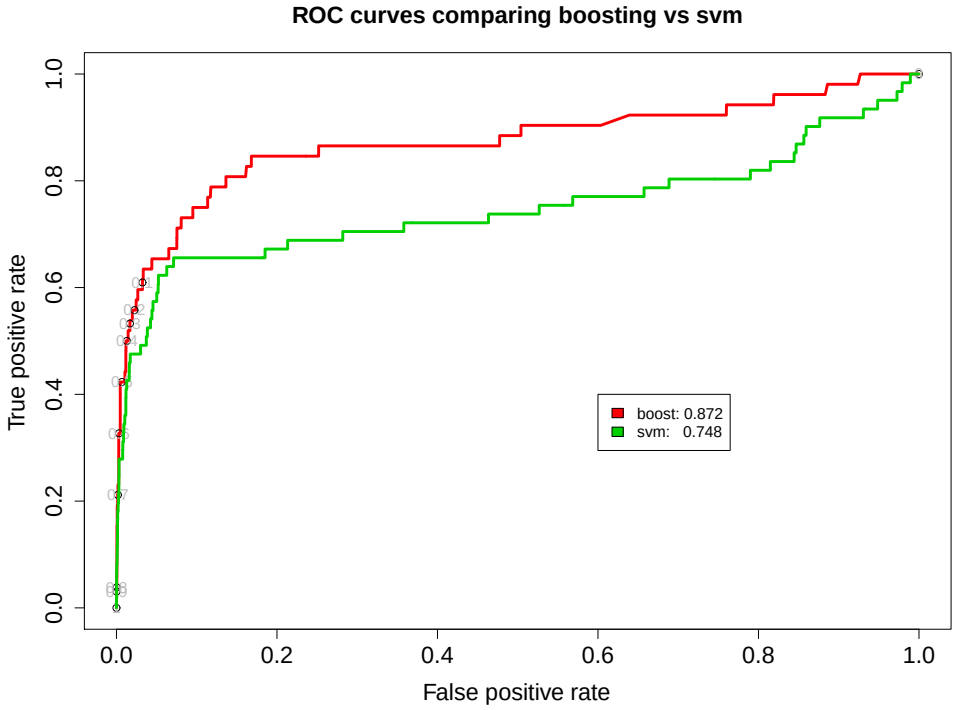


Figure 33: Comparison of ROC curves of boosting trees and support vector machine. The areas under ROC's apparently illustrates the advantages of boosting method over SVM. The results are based on wavelet-based definition of spikes.

The relative importance shown in Fig. 34 is another advantage of boosting trees: good interpretation of features. This quantity reveals the contribution of features in the prediction and serves as a feature selection scheme. In practice, hundreds of features may be available instead of the five features we considered in the current case study. Boosting trees will provide a guidance of variable selection, while SVM fails to deliver that information.

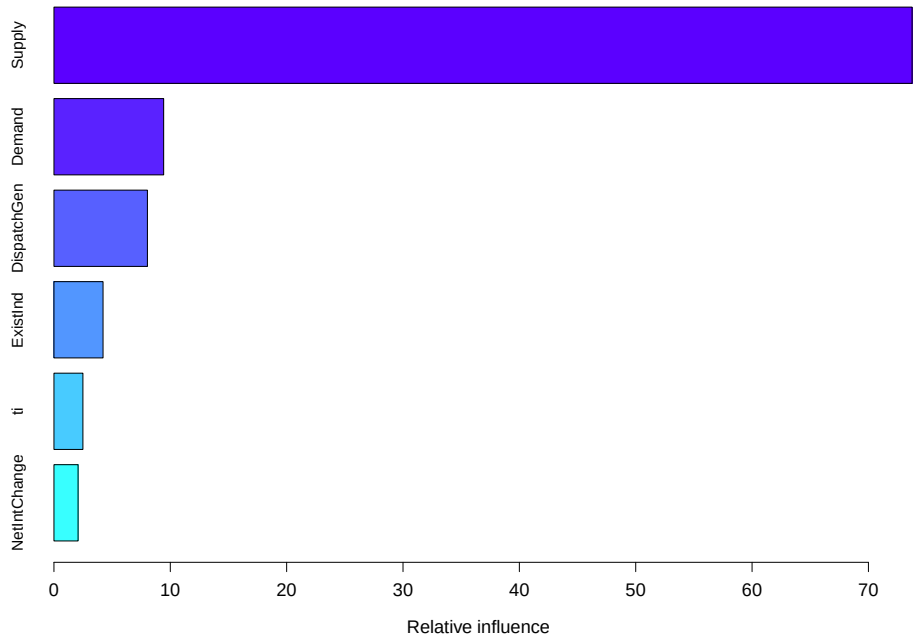


Figure 34: The relative importance of covariates in the boosting.

4.4.2 Price Spikes Prediction in New South Wales

The parameters are kept the same as those in Section 4.4.1. For instance, the maximum number of decision trees in boosting is set to be 4000. 33% of data is employed in the training and the remaining is used for testing purpose. The optimal tuning parameter, the number of decision trees, is determined through 5-fold cross-validation.

Table 5 summarizes the comparison of prediction between boosting trees and SVM. The first part of Table 5 shows the comparison relied on the wavelet-based definition, while the second part shows the results from hard-thresholding method. For both boosting trees and SVM, we report the true positive rate and false positive rate for spike definition with two different scenarios to illustrate the consistency of our comparison. The value of τ is the threshold for conditional probability in order to generate comparable error rates. In all comparisons, boosting tree dominates SVM in the sense that it has higher true positive rate and lower false alarm rate. Fig. 35 gives

Table 5: Summary of Numberical Experiments: NSW

Wavelet	6σ		8σ	
	Boosting	SVM	Boosting	SVM
Tpr	37.50%	26.70%	40.26%	27.30%
Fpr	2.17%	2.28%	1.41%	1.47%
τ	0.4	0.35	0.4	0.4
Hard	75		85	
	Boosting	SVM	Boosting	SVM
Tpr	46.80%	35.40%	46.80%	38.70%
Fpr	2.15%	2.78%	1.35%	1.40%
τ	0.4	0.5	0.5	0.35

the comparison of ROC curves between boosting trees and SVM. The area under the ROC curve verifies our observation as well.

4.5 Computational Efficiency

Since the usage of support vector machine highly depends on the accuracy of tuning parameter, the fine grid will be ideal for cross-validation in practice, which leads to intensive computation during the in-sample training period. For boosting trees, only the number of decision trees needs to be determined, and the model can be built extremely fast to adapt to the real-time spike prediction. The average computation time elapsed for generating prediction on the test data set is provided in Table 6. The learning algorithms are implemented in R and the computation is conducted on PC with Intel(R) Core(TM) i5-2520M CPU@2.50 GHz, 4.00 GB RAM and 64-bit Windows 7 operating system.

Table 6: Summary of Computation Time For Four Models

Time	BoostWave	BoostThresh	SVMWave	SVMThresh
Seconds	17.78	17.69	82.35	74.06

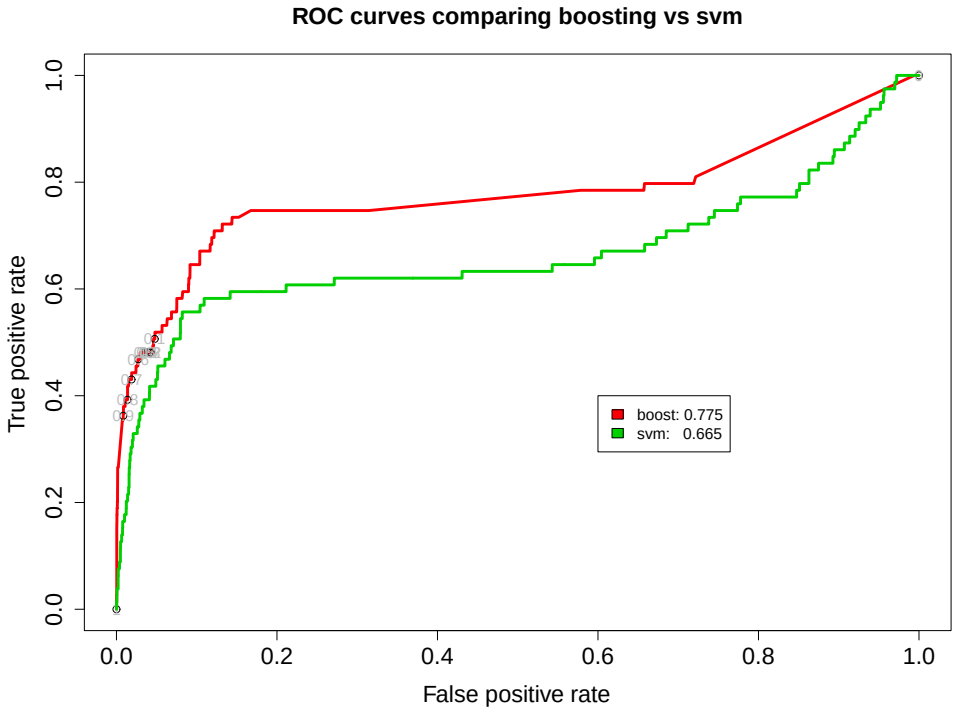


Figure 35: Comparison of ROC curves of boosting trees and support vector machine. The areas under ROC's apparently illustrates the advantages of boosting method over SVM. The results are based on hard-thresholding defintion of spikes.

4.6 Related Works and Further Extension

In machine learning community, other variants of boosting and support vector machine are proposed for imbalanced binary classification problem. The *RankBoost* proposed in [26] aims at maximization of area under ROC curve directly instead of minimization of classification errors and therefore it is more suitable for ranking problem. Recent work [62] shows that Adaboost and RankBoost have equally good asymptotic performance in binary classification and Adaboost achieves an area under ROC curve as good as RankBoost's. On the other hand, [60] provided a modification of SVM directly maximizing the area under ROC curve and shows that in separable case, these two variants are equivalent and in nonseparable case, their performance are equally good. Notice that since the classical SVM directly estimates the decision function, we employ the method introduced in [52] to convert the decision values into conditional probabilities.

For further development, the proposed methodology can be combined with manifold learning as in [9] for the modeling and prediction of electricity prices, incorporating both the long-term and short-term price dynamics. This could be quite valuable for risk management in the electricity market. Besides the choice of threshold for conditional probability in binary classifier can be formulated as optimization problem if the cost related to each type of misclassification is known. More specifically, the total cost is represented as follows.

$$\text{cost} = c(-|+)(1 - \text{Tpr}(\tau)) \cdot \text{Pos} + c(+|-)\text{Fpr}(\tau) \cdot \text{Neg}.$$

where $c(-|+)$ and $c(+|-)$ indicate the cost associated to the false negative and false positive respectively. Pos and Neg denotes the proportion of positives and negatives in the data. The optimal τ should be the one that minimizes the cost.

4.7 Conclusion

Detection and prediction of price spikes in electricity markets is an essential task for both trading and risk management. In this paper, we propose a new methodology which combines a gradient boosting trees method and nondecimated wavelet analysis to detect and forecast price spikes in real-time markets. Computational results in case studies demonstrate that this method outperforms the existing methodology in the literature. Nondecimated wavelet decomposition separates the trend and seasonality effects in the hourly electricity spot prices from the spike effects. It offers practitioners with a more systematic and precise approach to identify spikes. Moreover, the gradient boosting method inherits the good properties of decision trees such as robustness to the irrelevant covariates, fast computational capability and ease of interpretation, which other classification methods such as SVM do not have. Real-world data analysis shows that our proposed methodology yields significant improvement in prediction accuracy.

REFERENCES

[1] Agmon, S., *Lectures on Elliptic Boundary Value Problems*. Princeton, 1965.

[2] Arias-Castro, E., Donoho, D., and Huo, X., "Near-optimal detection of geometric objects by fast multiscale methods," *IEEE Trans. Information Theory*, vol. 51, no. 7, pp. 2402–2425, 2005.

[3] Behnke, S., *Local Multiresolutin Path Planning*, vol. 3020 of *Lecture Notes in Computer Science*, pp. 332–343. Berlin: Springer, 2004.

[4] Belkin, M. and Niyogi, P., "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, pp. 1373–1396, June 2003.

[5] Belkin, M. and Niyogi, P., "Towards a theorectical foundation for laplacian-based manifold methods," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1289–1308, 2008.

[6] Botea, A., Muller, M., and Schaeffer, J., "Near-optimal hierarchical pathfinding," *Journal of Game Development*, vol. 1, pp. 1–30, 2004.

[7] Bradley, A. P., "The use of the area under roc curve in evaluation of machine learning algorithm," *Pattern Recognition*, vol. 7, no. 30, 1997.

[8] Bulitko, V., Sturtevant, N., Lu, J., and Yau, T., "Graph abstraction in real-time heuristic search," *Journal of Artifical Intelligence Research*, vol. 30, pp. 51–100, 2007.

[9] Chen, J., Deng, S., and Huo, X., "Electricity price curve modeling and forecasting by manifold learning," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 877–887, 2008.

[10] Contreras, J., Espinola, R., Nogales, F. J., and Conejo, A. J., "Arima models to predict next-day electricity prices," *IEEE Trans on Power Systems*, vol. 18, pp. 1020–1040, Aug 2003.

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms*. Cambridge, MA: MIT Press and McGraw-Hill, 2nd ed., 2001.

[12] Cowlagi, R. and Tsiotras, P., "Multiresolution path planning with wavelets: A local replanning approach," in *American Control Conference*, (Seattle, WA), pp. 1220–1225, June 1-13 2008.

[13] Cowlagi, R. and Tsiotras, P., "Shortest distance problems in graphs using history-dependent transition costs with application to kinodynamic path planning," in *American Control Conference*, (St. Louis, MO), pp. 414–419, June 10–12, 2009.

[14] de Champeaux, D. and Sint, L., "An improved bidirectional heuristic search algorithm," *Journal of the ACM*, vol. 24, no. 2, pp. 177–191, 1977.

[15] Dijkstra, E., "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[16] Donoho, D. and Huo, X., "Beamlets pyramids: A new form of multiresolution analysis, suited for extracting lines, curves and objects from very noise image data," in *Proceedings of SPIE*, vol. 4119, pp. 434–444, July 2000.

[17] Donoho, D. and Huo, X., "Applications of beamlets to detection and extraction of lines, curves, and objects in very noisy images," in *Proceedings of Nonlinear Signal and Image Processing*, June 2001.

[18] Donoho, D. and Huo, X., *Multiscale and Multiresolution Methods*, vol. 20, ch. Beamlets and multiscale image analysis, pp. 149–196. Spring, 2002.

[19] Donoho, D. and Huo, X., "Beamlab and reproducible research," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 2, no. 4, pp. 391–414, 2004.

[20] Donoho, D. and Johnstone, I., "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455, 1994.

[21] Duchon, J., "Splines minimizing rotation invariant seminorms in sobolev spaces," *Constructive Theory of Functions of Several Variables*, vol. Berlin:Springer-Verlag, pp. 85–100, 1977.

[22] Felner, A., Sturtevant, N., and Schaeffer, J., "Abstraction-based heuristics with true distance computations," *Proceedings of the Eighth Symposium on Abstraction, Reformulatoin, and Approximation*, 2009.

[23] Ferguson, D., Likhachev, M., and Stentz, T., "A guide to heuristic-based path planning," in *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[24] Ferguson, D. and Stentz, T., "The field D* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," Tech. Rep. CMU-RI-TR-05-19, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, June 2005.

[25] Fredman, M. and Tarjan, R., "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[26] Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y., "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, no. 1, 2003.

[27] Freund, Y. and Schapire, R. E., "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 1, no. 55, 1997.

[28] Friedman, J. H., "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[29] Friedman, J. H., "Stochastic gradient boosting," *Computational Statistics and Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[30] Geisberger, R., "Contraction hierarchies: Faster and simpler hierarchical routing in road networks." Diploma Thesis, Institut für Theoretische Informatik, Universität Karlsruhe, 2008.

[31] Goldberg, A. and Harrelson, C., "Computing the shortest path: A* search meets graph theory," *In: 16th ACM-SIAM Symposium on Discrete Algorithms*, pp. 156–165, 2005.

[32] Goldberg, A., Kaplan, H., and Werneck, R., "Reach for A*: Efficient point-to-point shortest path algorithms," *Workshop on Algorithm Engineering and Experiments*, pp. 129–143, 2006.

[33] Goldenberg, M., Felner, A., Sturtevant, N., and Schaeffer, J., "Portal-Based True-Distance Heuristics for Path Finding," in *Third Annual Symposium on Combinatorial Search*, 2010.

[34] Guan, X., Ho, Y., and Pepyne, D. L., "Gaming and price spikes in electric power markets," *IEEE Transactions On Power Systems*, vol. 16, pp. 402–408, Aug. 2001.

[35] Guo, J. J. and Luh, P. B., "Selecting input factors for clusters of gaussian radial basis function networks to improve market clearing price prediction," *IEEE Trans on Power Systems*, vol. 18, no. 2, pp. 665–672, 2003.

[36] Harabor, D. and Botea, A., "Breaking Path Symmetries on 4-connected Grid Maps," in *Sixth Annual International AIIDE Conference*, (Palo Alto, CA), Oct. 11–13 2010.

[37] Hart, P., Nilsson, N., and Rafael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE trans. Sys. Sci. and Cyb.*, vol. 4, pp. 100–107, 1968.

[38] Hart, P. E., Nillsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics SSC4*, vol. 4, no. 2, pp. 100–107, 1968.

[39] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[40] Howard, P. G., Kossentini, F., Martins, B., Forchhammer, S., and Rucklidge, W. J., "The emerging JBIG2 standard," *IEEE Transactions On Circuits And Systems For Video Technology*, vol. 8, no. 7, pp. 838–848, 1998.

[41] Huo, X., *Sparse Image Representation via Combined Transforms*. PhD thesis, Stanford University, Department of Statistics, August 1999.

[42] Huo, X. and Chen, J., "JBEAM: multiscale curve coding via beamlets," *IEEE Trans. Image Processing*, vol. 14, pp. 1665–1677, Nov. 2005.

[43] Jansen, M. R. and Buro, M., "HPA* enhancements," in *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*, (Stanford, CA), pp. 84–87, The AAAI Press, June 2007.

[44] Johnson, D., "Efficient algorithms for shortest paths in sparse networks," *Journal of the ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[45] Kambhampati, S. and Davis, L. S., "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, vol. 2, no. 3, pp. 135–145, 1986.

[46] Koenig, S. and Likhachev, M., "D* lite," *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, pp. 476–483, 2002.

[47] Koenig, S., Likhachev, M., and Furcy, D., "Lifelong planning A*," *Artificial Intelligence Journal*, vol. 155, no. 1-2, pp. 93–146, 2004.

[48] Koenig, S., Likhachev, M., Liu, Y., and Furcy, D., "Incremental heuristic search in artificial intelligence," *Artificial Intelligence Magazine*, vol. 25, no. 2, pp. 99–112, 2004.

[49] Larsen, B., Burns, E., Ruml, W., and Holte, R., "Searching without a heuristic: Efficient use of abstraction," in *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pp. 114–120, 2010.

[50] LaValle, S. M., *Planning Algorithms*. Cambridge University Press, 2006.

[51] Li, K. C., "From stein's unbiased risk estimates to the method of generalized cross validation," *The Annals of Statistics*, vol. 13, pp. 1352–1377, 1985.

[52] Lin, H. T., Lin, C. J., and Wang, R. C., "A note on platt's probabilistic outputs for support vector machines," *Machine Learning*, vol. 68, no. 3, pp. 267–276, 2007.

[53] Lu, Y., Huo, X., Arslan, O., and Tsiotras, P., "An incremental, multi-scale search algorithm for dynamic path planning with low worst case complexity," *IEEE Transactions on Man, Systems and Cybernetics*, 2010. (submitted).

[54] Lu, Y., Huo, X., and Tsiotras, P., "Beamlet-like data processing for accelerated path-planning using multiscale information of the environment," in *The 49th IEEE Conference on Decision and Control*, (Hilton Atlanta Hotel, Atlanta, GA, USA), Dec 2010.

[55] Mount, T. D., Ning, Y., and Cai, X., "Predicting price spikes in electricity markets using a regime-switching model with time-varying parameters," *Energy Economics*, vol. 28, pp. 62–80, 2006.

[56] Nash, A., Daniel, K., Koenig, S., and Felner, A., "Theta*: Any-angle path planning on grids," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1177–1183, 2007.

[57] Pai, D. and Reissell, L.-M., "Multiresolution rough terrain motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 19–33, 1998.

[58] Percival, D. B. and Walden, A. T., *Wavelet Methods for Time Series Analysis*. Cambridge University Press.

[59] Pohl, I., *Machine Intelligence*, vol. 6, ch. Bi-directional Search, pp. 127–140. Edinburgh University Press, 1971.

[60] Rakotomamonjy, A., "Support vector machines and area under roc curves," tech. rep., 2004.

[61] Ramsay, T., "Spline smoothing over difficult regions," *Journal of the Royal Statistical Society: Series B(Statistical Methodology)*, vol. 64, pp. 307–319, 2002.

[62] Rudin, C. and Schapire, R. E., "Margin-based ranking and an equivalence between adaboost and rankboost," *Journal of Machine Learning Research*, vol. 10, no. 1, 2009.

[63] Samet, H., "Neighbor finding techniques for image represented by quadtrees," *Computer Graphics and Image Processing*, vol. 18, pp. 37–57, 1982.

[64] Sansom, D. and Saha, T. K., "Neural networks for forecasting electricity pool price in a deregulated electricity supply industry," in *the 1999 Australian Unverisities Power Engineering Conference*, (Darwin, Australian), pp. 214–219, Sep. 1999.

[65] Stentz, A., "Optimal and efficient path planning for unknown and dynamic environments," *International Journal of Robotics & Automation*, vol. 10, no. 3, pp. 89–100, 1995.

[66] Stone, C. J., "Optimal global rates of convergence for nonparamteric regression," *The Annals of Statistics*, vol. 10, pp. 1040–1053, 1982.

[67] Sturtevant, N. and Buro, M., "Partial pathfinding using map abstraction and refinement," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, p. 1392, 2005.

[68] Sturtevant, N., Felner, A., Barer, M., Schaeffer, J., and Burch, N., "Memory-based heuristics for explicit state spaces," in *International Joint Conference on Artificial Intelligence*, (Pasadena, CA), July 11–17 2009.

[69] Sturtevant, N. and Jansen, R., "An analysis of map-based abstraction and refinement," in *Proceedings of the 7th International conference on Abstraction, reformulation, and approximation*, pp. 344–358, Springer-Verlag, 2007.

[70] Sturtevant, N. and Geisberger, R., "A Comparison of High-Level Approaches for Speeding Up Pathfinding," in *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, (Palo Alto, CA), Oct. 11-13 2010.

[71] Trück, S., Weron, R., and Wolff, R., "Outlier treatment and robust approaches for modeling electricity spot prices," in *The 56th Session of the International Statistical Institute, Invited Paper Meeting IPM71 Statistics of risk aversion*, (Lisbon), Aug. 22-29 2007.

[72] Tsiotras, P. and Bakolas, E., "A hierarchical on-line path-planning scheme using wavelets," *European Control Conference*, July 2-5 2007.

[73] Utreras, F. I., "Convergence rates for multivariate smoothing spline functions," *Journal of Approximation Theory*, vol. 52, pp. 1–27, 1988.

[74] Wahba, G., *Spline Models for Observational Data*, vol. 59 of *CBMS-NSE Regional Conference Series in Applied Mathematics*. Philadelphia: SIAM, 1990.

[75] Wood, S. N., Bravington, M. V., and Hedley, S. L., "Soap film smoothing," *Journal Of The Royal Statistical Society Series B*, vol. 70, no. 5, pp. 931–955, 2008.

[76] Yahja, A., Stentz, A., Singh, S., and Brumit, B. L., "Framed-quadtree path planning for mobile robots operating in sparse environments," in *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, (Leuven, Belgium), pp. 650–655, IEEE, May 1998.

[77] Zhao, J. H., Dong, Z. Y., Li, X., and Wong, K. P., "A framework for electricity price spike analysis with advanced data mining methods," *IEEE Transactions on Power Systems*, vol. 22, no. 1, 2007.

[78] Zhou, X. and Belkin, M., "Semi-supervised learning by higher order regularization," *14th International Conference on Artificial Intelligence and Statistics*, vol. 15, 2011.